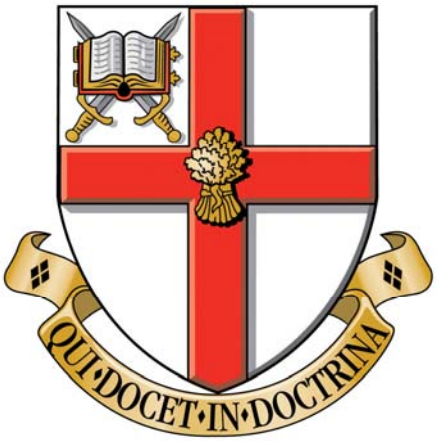


University of Chester

Author(s): Alexander John Martyn Little

Title: Analytical and numerical solution methods for integral equations using Maple symbolic algebraic package

Date: December 1998

Originally published as: University of Liverpool MSc dissertation

ANALYTICAL AND NUMERICAL SOLUTION METHODS FOR INTEGRAL EQUATIONS USING MAPLE SYMBOLIC ALGEBRAIC PACKAGE.

ALEXANDER JOHN MARTYN LITTLE.

Dissertation submitted to University College Chester for the Degree of Master of Science (Mathematics) in part fulfilment of the award.

Four Module Dissertation.

December 1998.

Abstract.

This dissertation considers the solution of Volterra integral equations of the form,

$$cf(t) = g(t) + \int_0^t K(t,s)f(s)ds, \qquad 0 \le t \le T < \infty,$$

where $c = 0$ or 1, and Fredholm integral equations of the form,

$$f(t) = g(t) + \int_a^b K(t,s)f(s)ds, \quad -\infty < a < b < \infty.$$

We consider analytical and numerical methods of solution, the theoretical basis for the methods, and present new solution routines in the Maple (release V2.0a) symbolic algebra package.

Declaration.

This work is original and has not been submitted previously in support of any qualification or course.

Signed .

Acknowledgements.

I would like to thank the European Social Fund for the part payment of fees and Dr. N. J. Ford and Dr. J. T. Edwards of University College Chester for their support and collaboration. I would also like to thank Paul Crofts B.Sc M.Phil for his proof reading of this thesis.

Contents.

Chapter 1.

Introduction.

Aim.

This thesis presents a sequence of programs developed in Maple to solve, analytically and numerically, wide classes of integral equations of Volterra and Fredholm types. Existing software (in NAG library, for example) concentrates on producing numerical solutions. Here we consider both numerical and analytical methods by appealing to Maple's symbolic algebra capabilities.

Differential and integral equations.

The use of differential equations for obtaining solutions to many natural and man-made phenomena, has been common practice for many years. However, there are situations where a differential equation cannot accurately represent the physical nature of the problem, and as a result the introduction of integral equations was realised. Although work on integral equations began in the 1820's with Abel, it is believed that Vito Volterra was the first who conceived the idea of a function whose behaviour depends on a continuous set of values obtained from another function, and his first paper on this subject was published in 1896. These integral equations of history dependant form, or as Volterra related 'materials with memory', are ideal for modelling physical and biological processes such as the Lotka-Volterra predator-prey equations. In this thesis we shall be concerned with the solution of linear Volterra integral equations of the first and second kind, and Fredholm integral equations of the second kind by means of implementing solution techniques using the Maple software package. The solution methods which will be used are as follows:

Analytical methods:-

- Conversion to ordinary differential equations.
- Laplace transform solution techniques.
- Resolvent kernel solution expression.

Numerical methods:-

- Neumann series.
- Trapezium method.
- Simpson's 3/8ths method.
- Gregory's method.
- Mid point rule.

i) For linear Volterra integral equations of the first kind with smooth kernels, where Richardson's extrapolation can be used.
ii) For Abel type equations, where Aitken's error correction term can be applied, also using Richardson's extrapolation formula.

The use of Laplace transforms will not be used in the treatment of Fredholm integral equations, (see section 4.3.1, chapter four). The analytical solution techniques mentioned above will be described in chapter four, whilst the numerical solution techniques will be covered in chapter five. The background theorems required for these solution methods are given in chapters two and three, where the existence and uniqueness of a solution is classified.

This chapter is intended as an introduction to integral equations and begins in §§1.1 and 1.2 where a classification for Volterra and Fredholm integral equations will be given. §1.3 will present some applications for integral equations, §1.4 will introduce the Maple software package and finally §1.5 presents an overview of the objective for this thesis.

1.1 Volterra equations.

Volterra integral equations are generally classed as either first or second kind. An equation of the form,

$$g(t) = \lambda \int_a^t K(t,s,f(s))ds,$$

is known as a Volterra integral equation of the first kind, where the unknown function *f* appears only under the integral sign. When an integral equation takes the form

$$f(t) = g(t) + \lambda \int_a^t K(t,s,f(s))ds, \qquad 0 \le t \le T.$$

it is called a Volterra equation of the second kind. The forcing function g(t) and the kernel K(t,s,f(s)) are assumed to be known, with the unknown being f(s). λ is a constant. The most common forms of Volterra integral equations have a lower bound on the integral of zero, and it is this case that will be used throughout. In order to solve the equation using a numerical method, it is necessary to restrict the upper limit of the integral *t*, to be less than infinity. A large upper limit does however produce a more comprehensive illustration to the long term behaviour of the solution, and so we will assume $0 \le t \le T < \infty$ unless otherwise stated.

In an equation where it can be seen that

$$K(t,s,(f(s)) = K(t,s)f(s) \qquad (1.1)$$

the equation is said to be linear. Conversely, a kernel of K(t,s,f(s)) is called non-linear if it cannot take the form of (1.1).

When the kernel is of the form k(t-s), the equation is called a convolution equation. Such equations can yield an explicit solution more readily by applying special analytical methods (Laplace transforms, Mikusiñski) which cannot be applied to kernels in (1.1), and consequently, this form of kernel is of particular interest in this thesis. Linear equations pose far fewer problems than non-linear cases where restrictions on the kernel are often required. As this thesis is primarily focused upon results obtained from a computer software package and not the complexity of the problem, it is *linear* Volterra equations of the second kind which will be modelled and analysed. Hence the equations which will be examined and tested will be of the form,

$$f(t) = g(t) + \lambda \int_0^t K(t,s) f(s) ds, \qquad 0 \le t \le T. \tag{1.2}$$

1.2 Fredholm integral equations.

Fredholm equations appear similar to Volterra equations, the only difference being the bounds on the integral. A Fredholm integral equation of the second kind is of the form,

$$f(t) = g(t) + \lambda \int_a^b K(t,s,f(s)) ds. \tag{1.3}$$

where $-\infty < a < b < \infty$.

Indeed, we can write a Volterra integral equation in the form of a Fredholm integral equation by defining the equation (1.2) as,

$$f(t) = g(t) + \lambda \int_0^T L(t,s) f(s) ds,$$

where

$$L(t,s) = \begin{cases} K(t,s), & 0 \le s \le t \\ 0, & s > t. \end{cases}$$

Again it is the linear form of equation (1.3) which will be studied, and we shall see in later chapters that the nature of the limits on the integral somewhat alter the technique for solution.

The main problem with Volterra and Fredholm integral equations is that all too often an explicit solution cannot be found. There are very few analytical methods and their application is generally limited to simple equations. Numerical techniques however, are able to produce an approximate solution to a much wider range of equations and therefore both analytical and numerical solution methods will be applied in this research.

1.3 Applications of integral equations.

The application of integral equations can be found almost anywhere. In general, the best known areas of application are in history dependent environments, where the condition of the problem at time t is determined by results at previous times, as well as the present time. A very elementary example of this is to consider the life span of a light bulb and when a replacement is mandatory, as presented by Burton [8], pp. 113-114.

The life span of a light bulb is random event, and so we can define the probability density function that a light bulb of age t will fail as f(t). If we then consider the probability that the light bulb will fail at a particular instant in time, for example the interval $(t, t+\delta t)$, then the resulting probability will be that of $f(t-s)\delta t$, where s is the time at which the light bulb was installed. The fact of f(t) being a probability density function means that over an infinite time period the probability of failure is 1. Similarly, the probability of a light time bulb failing which was installed at time s, is also 1. This can be expressed as follows,

$$\int_s^\infty f(t-s) dt = 1.$$

To establish a history dependent equation we introduce another function which relates to the rate at which each light bulb is replaced. We shall call this probability density function y(t), which yields $y(t)\delta t$, when describing the probability that a replacement light bulb is required at time $(t, t+\delta)$. The function y(t) is given by the probability of the first failure plus the probability there was another failure at time (t-s) and whether a replacement was made at time s. This can be written as,

$$y(t)\delta t = f(t)\delta t + \int_0^t f(t-s)\delta t y(s) \quad ds.$$

and when $\delta t \to 0$,

$$y(t) = f(t) + \int_0^t f(t-s)y(s)ds. \tag{1.4}$$

It can be seen from this equation that the event of a light bulb needing replacement is dependent upon when previous bulbs were replaced. Equation (1.4) can also be seen to be a linear Volterra integral equation of the second kind of convolution type. This example is classically referred to as a *'renewal'* equation. Renewal equations are the most prevalent group of history dependent equations, and it was a similar type of equation on which Volterra based his population dynamics models.

Volterra integral equations of the second kind where the kernel is of convolution type as in (1.4), are one of the preferred forms of problem, as they readily yield a solution. We shall see in chapter four how convolution equations of this type can be solved using analytical techniques.

Other applications of integral equations of Volterra or Fredholm type occur in problems of heat conduction and heat diffusion, such as those encountered in nuclear reactor dynamics or industrial smelting. Many kinds of systems, including electrical circuitry and computer systems, can also be modelled by integral equations. This area of application is called systems theory. More examples of the application of integral equations can be found in Burton [8] pp. 115-123.

Before examining the aforementioned solution methods we shall state some fundamental theorems of Volterra (chapter 2) and Fredholm (chapter 3) integral equations necessary for the solution methods to hold. In conclusion to this chapter we detail the software package chosen to program the solution algorithms

1.4 The Maple (release V2.0a) software package.

Many mathematical software packages, including Matlab and Mathematica were considered as part of this research, but the Maple software package was chosen because it contains three built in routines (*int*, *dsolve* and *laplace*) that are essential for the application of the analytical solution methods.

- The *int* routine (and the inert form *Int)*, is ideal for the manipulation and evaluation of integrals, and this is ideal for converting an integral equation into its ordinary differential form.
- *dsolve* is a Maple routine for solving ordinary differential equations with or without initial values. Again, this is perfect for solving the equivalent ordinary differential equation of an integral equation.
- The *laplace* routine can of course be used for Laplace transforms, which form the core of one of the analytical solution methods presented in chapter four.

Maple contains a facility by which individual packages can be defined and saved, allowing easy access to any routine. This option enables two separate packages

to be created, one for Volterra equations and one for Fredholm equations. Each package contains solution methods, both analytical and numerical, along with specific help pages designed as an aid for each routine. The emphasis has been put on Volterra equations, (where solutions to equations of the first kind are also undertaken), because much work has been done in the field of Fredholm equations. In depth documentation for each routine can be found in appendix one (Volterra), and appendix two (Fredholm).

Chapter 2.

Properties of linear Volterra integral equations of the first and second kind.

Introduction.

In this chapter we shall be looking at some established theorems on linear Volterra equations of the first and second kind. In section 2.1 we shall be using the Picard iterative method to prove whether or not a Volterra integral equation has a solution, and if so determine whether it is unique. This theorem is important for the analytical and numerical solution techniques described later in chapters four and five, which will then be converted into computer algorithms in Maple. Section 2.2 expands on the Picard iterative approach and demonstrates a valuable way of expressing a solution in a resolvent kernel manner. This result is then used in chapter 4, §4.3.1, where Laplace transform techniques are applied to linear Volterra equations of the second kind whose forcing function, g(t), is of a complex nature. In the concluding section of this chapter we take a brief look at the properties of linear Volterra equations of the first kind.

2.1 Proving the existence and uniqueness of a solution.

Before attempting to solve an equation it is desirable to prove the existence and uniqueness of the solution. The most preferred approach in the case of linear Volterra equations of the second kind is the Picard method, which, although not as powerful as some continuation methods (Linz [15] p. 29), is regarded by many texts to be the standard. We shall begin by presenting the Picard iterative process, shown by Linz [15] pp. 29-31.

2.1.1 The Picard iteration.

Taking the linear Volterra equation of the second kind,

$$f(t) = g(t) + \int_0^t k(t,s)f(s)ds, \tag{2.1}$$

Picard's method of successive approximation uses the iteration,

$$f_n(t) = g(t) + \int_0^t k(t,s)f_{n-1}(s)ds, \quad n=0,1,2,..., \tag{2.2}$$

where $f_0(t) = g(t)$, when n=0.

Replacing n with (n-1) and then subtracting from (2.2), we obtain the following equation,

$$\left[f_n = g(t) + \int_0^t k(t,s) f_{n-1}(s)ds\right] \rightarrow \left[f_{n-1} = g(t) + \int_0^t k(t,s) f_{n-2}(s)ds\right]$$

$$f_n - f_{n-1} = \int_0^t k(t,s)(f_{n-1} - f_{n-2})ds. \tag{2.3}$$

For simplicity we shall substitute $\varphi_n(t) = f_n(t) - f_{n-1}(t)$, n=0,1,2,..., into equation (2.3). Thus,

$$\varphi_n(t) = \int_0^t k(t,s)\varphi_{n-1}(s)ds, \tag{2.4}$$

where

$$\varphi_0(t) = g(t).$$

Working with the previous substitution of $\varphi_n(t) = f_n(t) - f_{n-1}(t)$, n=0,1,2,..., we obtain this result,

$$\begin{aligned} f_n(t) &= \varphi_n(t) + f_{n-1}, \\ f_n(t) &= \varphi_n(t) + \varphi_{n-1} + f_{n-2}, \\ &\vdots \\ f_n(t) &= \sum_{i=1}^{n} \varphi_i(t) + f_0(t), \end{aligned}$$

Since $f_0(t) = g(t) = \varphi_0(t)$,

$$f_n(t) = \sum_{i=0}^{n} \varphi_i(t). \tag{2.5}$$

We can now use these results to prove the existence and uniqueness of a solution under the conditions that k(t,s) and g(t) are continuous. In order to prove the existence of the solution we must first chose G and K such that,

$$\begin{aligned} |g(t)| &\le G, \qquad 0 \le t \le T \\ |k(t,s)| &\le K, \quad 0 \le s \le t \le T, \end{aligned}$$

where $0 \le T < \infty$.

Using induction:-

$$\begin{aligned} |\varphi_0(t)| &= g(t), \\ &\le G. \\ |\varphi_1(t)| &= \int_0^t k(t,s)G ds, \\ &\le KGt. \\ |\varphi_2(t)| &= \int_0^t K(KGs)ds, \\ &\le \tfrac{1}{2}K^2Gt^2. \\ |\varphi_3(t)| &= \int_0^t K(\tfrac{1}{2}K^2Gs^2)ds, \\ &\le \tfrac{1}{6}K^3Gt^3. \\ &\vdots \\ |\varphi_{n-1}(t)| &= \int_0^t K(\tfrac{1}{(n-2)!}K^{n-2}Gs^{n-2})ds, \\ &\le \tfrac{1}{(n-1)!}K^{n-1}Gt^{n-1}. \end{aligned}$$

$$|\varphi_n(t)| = \int_0^t K(\tfrac{1}{(n-1)!}K^{n-1}Gs^{n-1})ds,$$
$$\leq \tfrac{1}{n!}K^nGt^n,$$

Therefore, $|\varphi_n(t)| \leq \dfrac{G(Kt)^n}{n!}$, $0 \leq t \leq T$, $n = 0,1,....$ Since this is true for n=0, it is also true for all n, and so the equation $f_n(t) = \sum_0^n \varphi_i(t)$, will converge.

Hence,

$$f(t) = \sum_0^\infty \varphi_i(t). \tag{2.6}$$

We can prove that this result (2.6) satisfies the equation by substituting it into our original equation (2.1). i.e.

$$\int_0^t k(t,s)f(s)ds = \int_0^t k(t,s)\sum_{i=0}^\infty \varphi_i(s)ds$$
$$= \sum_{i=0}^\infty \int_0^t k(t,s)\varphi_i(s)ds.$$

and since from 2.4, $\varphi_i(t) = \int_0^t k(t,s)\varphi_{i-1}(s)ds$, it then follows,

$$\sum_{i=0}^\infty \int_0^t k(t,s)\varphi_i(s)ds = \sum_{i=0}^\infty \varphi_{i+1}(t)$$
$$= \sum_{i=0}^\infty \varphi_i(t) - \varphi_0(t)$$
$$= \sum_{i=0}^\infty \varphi_i(t) - g(t)$$
$$= f(t) - g(t).$$
$$f(t) = g(t) + \int_0^t k(t,s)f(s)ds.$$

Thus we have proven that when f(t) is expressed as $\sum_{i=0}^\infty \varphi_i(t)$, the equation $f(t) = g(t) + \int_0^t k(t,s)f(s)ds$, is satisfied, and because each $\varphi_i(t)$ term is continuous it follows that f(t) exists and is continuous.

With these results we can now proceed to show that the solution is unique. First we assume that there is another solution to the equation, which, like f(t), is also continuous. We call this other solution $\tilde{f}(t)$, which then gives

$$f(t) - \tilde{f}(t) = \int_0^t k(t,s)(f(s) - \tilde{f}(s))ds.. \tag{2.7}$$

If we now choose K and B such that

$$|k(t,s)| \leq K \qquad 0 \leq t \leq s \leq T,$$
$$|f(t) - \tilde{f}(t)| \leq B \quad 0 \leq t \leq T.$$

where B is a constant, and using induction,

$$|f(t) - \tilde{f}(t)| = \int_0^t KC ds,$$
$$\leq KCt.$$

$$\left|f(t)-\tilde{f}(t)\right| = \int_0^t K(KCs)ds,$$
$$\leq \tfrac{1}{2}K^2Ct^2.$$
$$\vdots$$
$$\left|f(t)-\tilde{f}(t)\right| = \int_0^t K\left(\tfrac{1}{(n-1)!}K^{n-1}Cs^{n-1}\right)ds,$$
$$\leq B(Kt)^n / n!,$$
where $0 \leq t \leq T, \quad n = 0,1,2,....$

When $n \to \infty$, $\dfrac{B(Kt)^n}{n!} \to 0$, and so $f(t)-\tilde{f}(t)=0$. i.e. $f(t)=\tilde{f}(t)$. Hence, there is only one continuous solution.

2.2 Obtaining a solution using the resolvent kernel.

Solving Volterra equations using analytical methods is often a long and complicated process, but the use of the resolvent kernel eliminates much of the calculation. As an example of an analytical method we shall briefly look at the use of resolvent kernels. The resolvent kernel technique was first used by Volterra in the late nineteenth century. This involved differentiating a first kind integral equation to obtain a second kind integral equation in which the kernel and the forcing function were expressed as partial derivatives; then the solution to the initial first kind equation could be expressed as,

$$f(t) = g(t) + \int_0^t R(t,s)g(s)ds, \quad t \in I,$$

where R(t,s) is the resolvent kernel.

The method shown here of expressing a solution in terms of a resolvent kernel can be seen in Linz [15] pp. 35-37 and Burton [8] pp. 61-62. We shall now show how this method can be applied to Volterra equations of the second kind, starting by taking equation (2.4) and through induction, expressing it in the following manner.

$$\varphi_n(t) = \int_0^t k(t,s)\varphi_{n-1}(s)ds$$
$$\varphi_1(t) = \int_0^t k(t,s)g(s)ds \qquad \text{where } \varphi_0(t) = g(t),$$
$$\varphi_2(t) = \int_0^t k(t,s)\varphi_1(s)ds,$$
$$= \int_0^t k(t,s)\int_0^s k(s,\tau)g(\tau)d\tau ds,$$
$$= \int_0^t \int_\tau^t k(t,s)k(s,\tau)ds\,g(\tau)d\tau,$$

and using the substitution $k_2(t,\tau) = \int_\tau^t k(t,s)k(s,\tau)ds$, we see that,

$$\varphi_2(t) = \int_0^t k_2(t,\tau)g(\tau)d\tau.$$

Similarly,

$$\varphi_3(t) = \int_0^t k(t,s)\varphi_2(s)ds,$$
$$= \int_0^t k(t,s)\int_0^s k_2(s,\tau)g(\tau)d\tau ds,$$
$$= \int_0^t \int_\tau^t k(t,s)k_2(s,\tau)ds\,g(\tau)d\tau,$$

substituting $k_3(t,\tau) = \int_\tau^t k(t,s)k_2(s,\tau)ds$,

$$\varphi_3(t) = \int_0^t k_3(t,\tau)g(\tau)d\tau.$$

Hence, we can see that

$$\varphi_n(t) = \int_0^t k_n(t,\tau)g(\tau)d\tau,$$

where $k_n(t,\tau) = \int_\tau^t k(t,s)k_{n-1}(s,\tau)ds$.

$$\text{for example,} \quad \varphi_n(t) = \int_0^t k_n(t,s)g(s)ds, \tag{2.8}$$

$$\text{where } k_n(t,s) = \int_s^t k(t,\tau)k_{n-1}(\tau,s)d\tau. \tag{2.9}$$

Using these newly derived equations and again working with Picard's iteration, we can develop an expression for the resolvent kernel. The first step involves taking the expression for the iterated kernels and using induction to obtain a new expression.

Taking the equation,

$$k_n(t,s) = \int_s^t k(t,\tau)k_{n-1}(\tau,s)d\tau.$$

and assuming that k(t,s) is continuous, with $k_1(t,s) = k(t,s)$, and that the inequality $|k(t,s)| \le K$, where $0 \le s \le t \le T$ holds, then by applying induction,

$$|k_1(t,s)| \le K,$$

$$|k_2(t,s)| \le \int_s^t k(t,\tau)K d\tau,$$

since $|k(t,s)| \le K$, $\therefore |k(t,\tau)| \le K$,

$$\le \int_s^t K^2 d\tau,$$

$$\le K^2(t-s).$$

$$|k_3(t,s)| \le \int_s^t k(t,\tau)K^2(\tau-s)d\tau,$$

$$\le \int_s^t K^3(\tau-s)d\tau,$$

$$\le K^3(t-s)^2/2.$$

Similarly,

$$|k_4(t,s)| \le \int_s^t k(t,\tau)\left(K^3(\tau-s)^2/2\right)d\tau,$$

$$\le \int_s^t \left(K^4(\tau-s)^2/2\right)d\tau,$$

$$\le K^4(t-s)^3/6.$$

Hence, we can see that,

$$|k_n(t,s)| \le \frac{K^n(t-s)^{n-1}}{(n-1)!}.$$

Secondly we take another equation (2.5) from the Picard approximation method and combine it with (2.8) as follows,

$$f_n(t) = \sum_{i=0}^{n} \varphi_i(t),$$

$$f_n(t) = \sum_{i=0}^{n} \int_0^t k_i(t,s)g(s)ds,$$

$$f_n(t) = g(t) + \sum_{i=1}^{n} \int_0^t k_i(t,s)g(s)ds,$$

$$f_n(t) = g(t) + \int_0^t \sum_{i=1}^{n} k_i(t,s)g(s)ds,$$

introducing the substitution $R_n(t,s) = \sum_{i=1}^{n} k_i(t,s)$, we arrive at,

$$f_n(t) = g(t) + \int_0^t R_n(t,s)g(s)ds.$$

This equation can then be used to express the solution to a linear Volterra equation of the second kind. The theory of the resolvent kernel expression is tested in practice in chapter four, §4.3.1, where Laplace transforms are used.

2.3 Linear Volterra integral equations of the first kind.

Equations of the first kind may be classed as either Abel or non Abel type. Abel equations are generally defined as those containing singularities; that is to say, the kernel is unbounded at s=t. Such equations are (in analytical terms) either complex to solve or lack the theorems necessary to do so. For this reason we shall be looking at non Abel equations which are generally classed as equations with smooth kernels.

The linear Volterra integral equation of the first kind,

$$g(t) = \int_0^t K(t,s)f(s)ds, \quad 0 \le t \le T. \tag{2.10}$$

which possesses a smooth kernel, has a solution if g(0)=0. The theorem for proving this solution is unique and continuous firstly converts (2.10) into a Volterra equation of the second kind by differentiation. Thus (2.10) becomes

$$g'(t) = \int_0^t \frac{\partial}{\partial t} K(t,s)f(s)ds + K(t,t)f(t).$$

$$f(t) = \frac{g'(t)}{K(t,t)} - \int_0^t \frac{1}{K(t,t)} \frac{\partial}{\partial t} K(t,s)f(s)ds. \tag{2.11}$$

and under the assumptions,

i). $K(t,s), \frac{\partial}{\partial t}K(t,s)$ exists and is continuous for $0 \le s \le t \le T$.

ii). $K(t,t) \neq 0$.

iii). $g(0)=0$.

iv). $g(t), g'(t)$ are continuous for $0 \le t \le T$.

then (2.10) possesses a unique continuous solution.

Proof: Taking equation (2.11) and making it analogous to the linear Volterra equation of the second kind,

$$f(t) = L(t) - \int_0^t H(t,s)f(s)ds, \quad 0 \le t \le T. \tag{2.12}$$

where,

$$L(t) = \frac{g'(t)}{K(t,t)}, \quad H(t,s) = \frac{1}{K(t,t)} \frac{\partial}{\partial t} K(t,s).$$

Equation (2.12) can then be proven to possess a unique continuous solution using Picard's method, as described earlier. Hence, for equation (2.10) there exists a unique continuous solution f(t).

The affinity of first kind equations to those of the second kind, means that the same solution techniques can be applied, albeit in a slightly modified way. This is true only for analytical methods. The numerical solution of linear Volterra equations of the first kind will be analysed in chapter five, §5.3.

Chapter Three.

Properties of linear Fredholm integral equations of the second kind.

In this chapter we will be introducing the theorems for proving the existence and uniqueness of a solution to a linear Fredholm integral equation of the second kind, as shown by Courant and Hilbert [12], pp. 116-118, and Kress [14], p. 45. The theorems discussed are important, as they form the basis upon which the solution methods described in chapters four and five are constructed. As the majority of equations which can be solved using these methods will have kernels of degenerate form, it is the theorems and proofs for such kernels which are given here. Theorems and proofs for other kernel types can be found in Courant and Hilbert [12], pp 119-122.

We shall begin by taking our linear Fredholm integral equation of the second kind,

$$f(t) - \lambda \int_a^b K(t,s) f(s) ds = g(t) \tag{3.1}$$

where both the forcing function and the kernel are continuous, the forcing function is integrable, the range of integration is finite, and define the kernel to be of degenerate form,

$$K(t,s) = \sum_{i=1}^{p} \alpha_i(t) \beta_i(t). \tag{3.2}$$

Then there are two possible solution outcomes.

i) (The first alternative.) There is a unique continuous solution, f(t), for any g(t), (in particular the solution f(t)=0), and the transposed equation of (3.1),

$$f(t) - \lambda \int_a^b K(s,t) f(s) ds = h(t), \tag{3.3}$$

also possesses a unique solution for every h(t).

ii) (The second alternative.) The homogeneous equation of (3.1),

$$\psi(t) = \lambda \int_a^b K(t,s) \psi(s) ds, \tag{3.4}$$

possesses a finite number of linearly independent solutions, $\psi(t_1), \psi(t_2), \ldots, \psi(t_n)$, and the transposed homogenous equation of (3.1),

$$\varphi(t) = \lambda \int_a^b K(s,t) \varphi(s) ds \tag{3.5}$$

also possesses a finite number of linearly independent solutions, $\varphi(t_1),\varphi(t_2),\ldots,\varphi(t_n)$. Then equation (3.1) has a solution if and only if the forcing function g(t) satisfies,

$$(g(t),\varphi_i(t)) = \int_a^b g(t)\varphi_i(t)dt = 0, \qquad (i = 1,2,\ldots,n).$$

This solution is only unique when

$$(f(t),h_i(t)) = \int_a^b f(t)h_i(t)dt = 0, \qquad (i = 1,2,\ldots,n).$$

Proof of first alternative.

Substituting (3.2) into (3.1), thus,

$$f(t) - \lambda\sum_{i=1}^{m}\alpha_i(t)\int_a^b\beta_i(s)f(s)ds = g(t), \tag{3.6}$$

and multiplying by $\beta_j(t)$,

$$\beta_j(t)f(t) - \lambda\sum_{i=1}^{m}\alpha_i(t)\beta_j(t)\int_a^b\beta_i(s)f(s)ds = \beta_j(t)g(t)$$

we can integrate with respect to *t*, to obtain,

$$\int_a^b\beta_j(t)f(t)dt - \lambda\sum_{i=1}^{m}\int_a^b\beta_j(t)\alpha_i(t)dt\int_a^b\beta_i(s)f(s)ds = \int_a^b\beta_j(t)g(t)dt,$$

which, when $c_{ji}(t) = (\beta_j(t),\alpha_i(t))$, and $x_i(s) = (\beta_i(s),f(s))$, gives

$$x_j(t) - \lambda\sum_{i=1}^{m}c_{ji}(t)x_i(s) = g_j(t), \tag{3.7}$$

for $(j = 1,2,\ldots,m)$.

If this system of equations possesses a unique solution, $x_1(t),x_2(t),\ldots,x_m(t)$, then the solution to equation (3.1) is given by,

$$f(t) = g(t) + \lambda\sum_{i=1}^{m}x_i(s)\alpha_i(t),$$

which may be verified by substituting into equation (3.6), thus

$$\left[g(t) + \lambda\sum_{i=1}^{m}x_i(s)\alpha_i(t)\right] - \lambda\sum_{i=1}^{m}\alpha_i(t)\int_a^b\beta_i(s)f(s)ds = g(t),$$

where $x_i(s) = (\beta_i(s),f(s))$.

Likewise, the transposed equation (3.3) can be turned into a system of linear equations,

$$y_j(t) - \lambda\sum_{i=1}^{m}c_{ij}(t)y_i(s) = h_j(t), \qquad (j = 1,2,\ldots,m),$$

which has a unique solution , $y_1(t),y_2(t),\ldots,y_m(t)$, and so the transposed equation (3.3) has a solution of

$$f(t) = h(t) + \lambda\sum_{i=1}^{m}y_i(s)\beta_i(t).$$

Proof of second alternative.

The system of linear equations of the homogeneous equation (3.4), given by

$$x_j(t) = \lambda\sum_{i=1}^{m}c_{ij}x_i$$

possesses the solution $x_1(t),x_2(t),\ldots,x_m(t)$, and so two linearly independent solutions $x_1(t),x_2(t),\ldots,x_m(t)$, and $x'_1(t),x'_2(t),\ldots,x'_m(t)$, will produce two linearly independent solutions of (3.4). The existence of these solutions is alike to the

existence of an identical number, n, of linearly independent solutions of the transposed equation,

$$y_j(t) - \lambda \sum_{i=1}^{m} c_{ij}(t) y_i(s) = 0$$

(these solutions being $y_{k1}(t), y_{k2}(t), \ldots, y_{km}(t)$, k=1,2,...,n,), and therefore of an identical number of linearly independent solutions $\varphi_1(t), \varphi_2(t), \ldots, \varphi_n(t)$ to the transposed homogeneous equation (3.5), where

$$\varphi_k(t) = \lambda \sum_{i=1}^{m} y_{kj}(t) \beta_j(s).$$

Equations (3.7), and the transposed system of linear equations, always possess a unique solution for n=0. When n>0, (3.7) and therefore (3.1), will only possess a solution when

$$\sum_{i=1}^{m} g_j y_{kj} = 0, \quad \text{where } k=1,2,\ldots,n,$$

which is equivalent to

$$\int_a^b g(t) \varphi_k(t) dt = 0, \qquad \text{where } k=1,2,\ldots,n.$$

This theorem for the existence and uniqueness of a solution to a linear Fredholm integral equation of the second kind is required for the following solution methods such as the Neumann series to hold.

Chapter Four.

Analytical solution methods.

In this chapter we shall cover the main analytical solution techniques for linear Volterra and Fredholm integral equations. The first method we shall examine is the conversion to ordinary differential form and subsequent solution. This method of conversion can be applied to both Volterra and Fredholm equations, but the solution technique of the resulting ordinary differential equation is decidedly different. The second analytical solution method is from the application of Laplace transforms. This form of solution is only applicable to Volterra integral equations (see §4.3.1). The background theorems for proving whether a problem has a solution, have been previously described in chapters two and three. This chapter, and chapter five, present the solution techniques which will be implemented into a self-contained software package in Maple, the results of which can be seen in the appendices.

4.1.1 Conversion to ordinary differential equation form.

In special cases a Volterra or Fredholm integral equation can be solved by conversion to an ordinary differential equation. What properties an equation must contain for this process to succeed will be discussed later, but first we shall demonstrate the conversion method. Taking our standard linear Volterra integral equation of the second kind, thus,

$$f(t) = g(t) + \lambda \int_0^t K(t,s) f(s) ds, \tag{4.1}$$

and differentiating both sides with respect to t, (assuming g(t) and K(t,s) are differentiable),

$$f'(t) = g'(t) + \frac{d}{dt}\left(\lambda \int_0^t K(t,s) f(s) ds \right),$$

we apply Leibniz's rule of differentiation upon an integral (Abramowitz [2] p.11), to form,

$$f'(t) = g'(t) + \lambda\left(\int_0^t \frac{\partial}{\partial t}\{K(t,s) f(s)\} ds + K(t,t) f(t) \frac{dt}{dt} - K(0,t) f(0) \frac{d0}{dt} \right),$$

$$f'(t) = g'(t) + \lambda\left(\int_0^t \frac{\partial}{\partial t}\{K(t,s) f(s)\} ds + K(t,t) f(t) \right). \tag{4.2}$$

Rearranging equation (4.1), to be,

$$\int_0^t K(t,s) f(s) ds = \tfrac{1}{\lambda}\big(f(t) - g(t)\big), \tag{4.3}$$

and if

$$\int_0^t K(t,s)f(s)ds = \int_0^t \frac{\partial}{\partial t}\{K(t,s)f(s)\}ds, \qquad (4.4)$$

then we use the substitution (4.3) in (4.2) to give,

$$f'(t) = g'(t) + f(t) - g(t) + K(t,t)f(t).$$

This ordinary differential equation can then be solved to find a solution for f(t), where f(0)=g(0). If, however, (4.4) does not hold, the equation (4.3) can be differentiated again and the substitution made if, (assuming second derivative of K(t,s) exists),

$$\int_0^t K(t,s)f(s)ds = \int_0^t \frac{\partial^2}{\partial t^2}\{K(t,s)f(s)\}ds.$$

with initial values of f(0)=g(0), and $f'(0) = g'(0) + f(0) - g(0) + K(0,0)f(0)$.

In general a linear Volterra equation of the second kind may be converted to an ordinary differential equation and then solved if

$$\int_0^t K(t,s)f(s)ds = \int_0^t \frac{\partial^n}{\partial t^n}\{K(t,s)f(s)\}ds,$$

for some $n \in N$.

Example. Consider the equation,

$$f(t) = e^t + \lambda\int_0^t \sin(t-s)f(s)ds. \qquad (4.5)$$

Our substitution expression is $\int_0^t \sin(t-s)f(s)ds = \frac{1}{\lambda}(f(t) - e^t)$. (4.6)

Differentiation of (4.5) yields,

$$f'(t) = e^t + \lambda\left(\int_0^t \cos(t-s)f(s)ds + \sin(0)f(t)\right),$$

$$f'(t) = e^t + \lambda\left(\int_0^t \cos(t-s)f(s)ds\right).$$

Differentiating a second time gives,

$$f''(t) = e^t + \lambda\left(\cos(0)f(t) - \int_0^t \sin(t-s)f(s)ds\right),$$

using (4.6),

$$f''(t) = e^t + \lambda\left(f(t) - \left(\frac{1}{\lambda}\left(f(t) - e^t\right)\right)\right),$$

i.e., the equation (4.5) can be converted to the ordinary differential equation,

$$f''(t) = 2e^t + (\lambda - 1)f(t).$$

which has the solution (when $\lambda = 1$, f(0)=1, $f'(0) = 1$) of

$$f(t) = 2e^t - 1 - t.$$

4.1.2 Application to linear Volterra equations of the first kind.

The same technique of conversion to an ordinary differential equation can be applied to an equation of the first kind. Taking a linear Volterra equation of the first kind,

$$g(t) = \lambda \int_0^t K(t,s)f(s)ds,$$

differentiation gives,

$$g'(t) = \lambda \int_0^t \frac{\partial}{\partial t} K(t,s)f(s)ds + \lambda K(t,t)f(t).$$

Once again, if $\int_0^t K(t,s)f(s)ds = \int_0^t \frac{\partial}{\partial t}\{K(t,s)f(s)\}ds$, then our ordinary differential equation is,

$$g'(t) = g(t) + \lambda K(t,t)f(t),$$

where the solution is given by,

$$f(t) = \psi(t)(g'(t) - g(t)),$$

where $\psi(t) = \frac{1}{\lambda K(t,t)}$.

This solution method holds only when $k(t,t) \neq 0$.

If the differentiated equation of the first kind does not contain the original integrand but does contain the f(t) function, the equation can be repeatedly differentiated until a substitution is found. For example, take the equation

$$e^t - t - 1 = \int_0^t \cos(t-s)f(s)ds,$$

twice differentiation yields a substitution of g(t) for the integral term,

$$e^t = -e^t + t + 1 + \frac{d}{dt}f(t),$$

therefore $f(t) = 2e^t - \frac{1}{2}t^2 - t + C$, where $C \in \Re$.

As we have seen in this section, the solution of Volterra integral equations by the conversion to an ordinary differential equation can only be obtained for equations with special kernels. These kernels have the following properties:

1. For some $n \in N$, $\int_0^t K(t,s)f(s)ds = \alpha \int_0^t \frac{\partial^n}{\partial t^n}\{K(t,s)f(s)\}ds$, and K(t,s) is differentiable.

2. $K(t,t) \neq 0$.

For linear Volterra integral equations of the second kind only property (1) must be true. First kind equations will require both properties to be true.

4.2 Converting Fredholm integral equations to ordinary differential equations.

The method for conversion of Fredholm integral equations is essentially the same as described in §4.1. Dissimilarity only arises when Leibniz's rule is applied to the integral and the only remaining term is the integral itself. With Volterra equations the lower bound is always zero, thus eliminating part of the resulting differentiated integral. Fredholm equations, because their bounds are constant integers, become more simple to manipulate with the disappearance of more terms. Consider the standard linear Fredholm integral equation of the second kind,

$$f(t) = g(t) + \alpha \int_a^b K(t,s)f(s)ds. \qquad (4.7)$$

Differentiation yields,

$$f'(t) = g'(t) + \alpha\int_a^b \frac{\partial}{\partial t}K(t,s)f(s)ds + K(t,b)f(b)\frac{\partial b}{\partial t} - K(t,a)f(a)\frac{\partial a}{\partial t},$$

$$\therefore \quad f'(t) = g'(t) + \alpha\int_a^b \frac{\partial}{\partial t}K(t,s)f(s)ds, \tag{4.8}$$

and if

$$\frac{\partial}{\partial t}K(t,s)f(s) = \alpha K(t,s)f(s), \text{ where } \alpha \neq 0,$$

then from (4.7),

$$f'(t) = g'(t) + f(t) - g(t).$$

When this ordinary differential equation is solved there will be unknown constants in the answer. With Volterra equations, initial values can be obtained and used in the solution as seen in §4.1.1. The method used to find the constants for Fredholm integral equations involves obtaining simultaneous equations. For example, taking the linear Fredholm integral equation of the second kind, thus

$$f(t) = \cos(t) + \int_0^1 \cos(t-s)f(s)ds, \tag{4.9}$$

differentiation yields,

$$f'(t) = -\sin(t) - \int_0^1 \sin(t-s)f(s)ds, \tag{4.10}$$

$$f''(t) = -\cos(t) - \int_0^1 \cos(t-s)f(s)ds, \tag{4.11}$$

hence the equivalent ordinary differential equation is,

$$f''(t) = -f(t), \tag{4.12}$$

whose solution is

$$f(t) = A\sin(t) + B\cos(t). \tag{4.13}$$

The values of the constants A and B are found by obtaining equations for f(0) and f'(0). From 4.9,

$$f(0) = 1 + \int_0^1 \cos(-s)f(s)ds,$$

and substituting (4.13) we have,

$$f(0) = 1 + \int_0^1 \cos(-s)(A\sin(s) + B\cos(s))ds,$$

which can be evaluated to,

$$f(0) = 1 + 0.3540367091A + 0.7273243567B,$$

also (4.13) yields f(0)=B, so

$$B = 1 + 0.3540367091A + 0.7273243567B. \tag{4.14}$$

Similarly, from(4.10),

$$f'(0) = -\ 0.2726756433A - 0.3540367091B,$$

Differentiating (4.13) and letting t=0 gives, f'(0) = A,

$$\therefore \quad A = -\ 0.2726756433A - 0.3540367091B \tag{4.15}.$$

These two simultaneous equations, (4.14) and (4.15), give the solutions for A and B of, A=4.851037641 and B=9.965853091, which can be entered into (4.13) to give the final solution for f(t).

The simplification of second kind equations, when Leibniz's rule of differentiation is applied, eliminates the possibility of finding a solution to a first kind equation. When the first kind equation of Fredholm type,

$$g(t) = \alpha \int_a^b K(t,s) f(s) ds,$$

is differentiated, we obtain,

$$g'(t) = \alpha \int_a^b \frac{\partial}{\partial t} K(t,s) f(s) ds,$$

and if

$$\alpha \int_a^b K(t,s) f(s) ds = \alpha \int_a^b \frac{\partial}{\partial t} K(t,s) f(s) ds,$$

the following equation is realised,

$$g'(t) = g(t),$$

which, when attempting to obtain a solution for f(t) (that does not exist in the above equation,) is obviously futile.

4.3.1 Laplace transform solution techniques.

As we have seen in §2.2, using iterated kernels as a way of expressing the resolvent kernel would be a very complex method. If the kernel is of a simple convolution type then the preferred approach is to apply Laplace transforms using the following convolution theorem, (Bolton [6] p.33), :-

$$\ell\{K(t) * f(t)\} = \int_0^t K(t-s) f(s) ds.$$

where $\ell\{f(t)\} = \int_0^\infty e^{-\omega t} f(t) dt$, and ω is a complex number. The result is the function being changed from the 'time domain' into the ω- domain. The Laplace transform integral has a range from 0 to infinity, and since Fredholm integrals are not of this type, Laplace transforms cannot be applied. In Volterra convolution equations where the forcing function is simple, Laplace transforms can be used on the entire equation to produce a solution. Conversely, equations with complex forcing functions can have Laplace transforms applied only to the functions under the integral sign and the solution expressed in terms of a resolvent kernel. Taking our former conditions of a simple forcing function in a second order linear equation of convolution type,

$$f(t) = g(t) + \int_0^t K(t-s) f(s) ds,$$

and by applying Laplace transforms we find an expression for the solution as follows,

$$\ell\{f(t)\} = \ell\{g(t)\} + \ell\{K(t) * f(t)\},$$

and since $\ell\{K(t) * f(t)\} = \ell\{K(t)\}\ell\{f(t)\}$, then,

$$\ell\{f(t)\} = \frac{\ell\{g(t)\}}{1 - \ell\{K(t)\}}.$$

Finally, by taking the inverse Laplace transforms we arrive at a solution expression thus,

$$f(t) = \ell^{-1}\left\{ \frac{\ell\{g(t)\}}{1 - \ell\{K(t)\}} \right\}.$$

In the case where a complex forcing function is present, the same method as above is applied to the resolvent kernel equation,

$$R(t) = K(t) + \int_0^t K(t-s) R(s) ds,$$

where the solution $R(t) = \ell^{-1}\left\{\dfrac{\ell\{K(t)\}}{1-\ell\{K(t)\}}\right\}$, is inserted in the equation,

$$f(t) = g(t) + R(t) * g(s),$$

giving the solution expression as,

$$f(t) = g(t) + \int_0^t R(t-s)g(s)ds.$$

Example. In the equation,

$$f(t) = g(t) + \int_0^t e^{(t-s)}f(s)ds,$$

g(t) is a complex function, therefore using the resolvent kernel approach our expression for R(t) is,

$$R(t) = \ell^{-1}\left\{\frac{\ell\{e^t\}}{1-\ell\{e^t\}}\right\},$$

$$= \ell^{-1}\left\{\frac{1}{(w-2)}\right\},$$

$$= e^{2t}$$

Hence the solution expressed as a resolvent kernel function is,

$$f(t) = g(t) + \int_0^t e^{2(t-s)}g(s)ds.$$

4.3.2 First order Volterra integral equations.

Laplace transforms can also be applied to Volterra integral equations of the first kind. The technique is, as above, very straightforward. For example, the first order equation

$$\int_0^t K(t-s)f(s)ds = g(t),$$

has the solution,

$$f(t) = \ell^{-1}\left\{\frac{\ell(g(t)}{\ell(K(t)}\right\}.$$

Example.

Solve the convolution Volterra integral equation of the first kind,

$$\int_0^t \sin(t-s)f(s) = t^2 + 2\cos(t) - 2.$$

$$\ell\{(t^2 + 2\cos(t) - 2)\} = \ell\{t^2\} + \ell\{2\cos(t)\} - \ell\{2\}$$

$$= \frac{2}{w^3} + \frac{2w}{w^2+1} - \frac{2}{w}$$

$$= \frac{2}{w^5 + w^3}$$

$$\ell\{\sin(t)\} = \frac{1}{w^2+1}$$

$$\frac{\ell\{g(t)\}}{\ell\{\sin(t)\}} = \frac{2}{w^3}$$

$$f(t) = \ell^{-1}\left\{\frac{2}{w^3}\right\}$$

$$f(t) = t^2.$$

The solution can now be checked by substituting $f(t) = t^2$ into the original equation thus,

$$\int_0^t \sin(t-s)s^2 ds = g(t)$$

integrating by parts we obtain,

$$= t^2 - 2\int_0^t s\cos(t-s)ds$$

$$= t^2 - 2\int_0^t \sin(t-s)ds$$

$$= t^2 - 2 + 2\cos(t)$$

$$= g(t).$$

This method of verifying the solution will be incorporated into the software solution routine (see appendix one). The methods we have seen in this chapter provide solutions to only a limited number of integral equations, whose properties have been listed previously. This is often the case where a solution is required for an equation that does not have such properties and so numerical methods are the favoured solution format.

Chapter 5.

Numerical solution methods for Volterra and Fredholm integral equations.

In the last chapter we saw how certain equations may be solved analytically, and outlined the limitations of these methods. Very few practical Volterra and Fredholm integral problems can be solved analytically, often as a result of insufficient solution methods. We shall now see how iterative and numerical methods can be applied. The first of these is the Neumann series (§5.1). In §§5.2 and 5.3, examine the numerical approaches to Volterra equations; while in §5.4 Fredholm equations are studied. It is these numerical methods which will be coded in the Maple package for the solution of integral equations.

5.1 The Neumann series of solution.

The Neumann series follows on from work we have previously looked at in §§2.1 and 2.2, involving iterated kernels. Although it may be classed as both an analytical and numerical method, its use here as an evaluation series allows us to place it in the numerical category. It was in 1877 that Neumann finalised the convergence theorems relating to the iterated kernels and although his work related to Fredholm integral equations, the same techniques can be applied to Volterra equations.

We take the standard linear Volterra equation of the second kind (2.1), and introduce a linear operator, $\boldsymbol{k}$, thus,

$$\boldsymbol{k}f = \int_a^b k(t,s)f(s)ds,$$

and the following sequence is obtained,

$$\begin{aligned} f_1(t) &= g(t). \\ f_2(t) &= g(t) + \boldsymbol{k}f_1(t), \\ &= g + \boldsymbol{k}g. \\ f_3(t) &= g + \boldsymbol{k}f_2(t), \\ &= g + \boldsymbol{k}(g + \boldsymbol{k}g), \\ &= g + \boldsymbol{k}g + \boldsymbol{k}^2 g. \\ &\vdots \\ f_n(t) &= g(t) + \boldsymbol{k}f_{n-1}(t), \\ &= g + \boldsymbol{k}g + \boldsymbol{k}^2 g + \boldsymbol{k}^3 g + \ldots + \boldsymbol{k}^{n-1} g. \end{aligned}$$

If a sufficient number of steps are taken then the sequence will hopefully converge to the function f(t). If the iteratives do not converge then a solution cannot be found. A working example helps to illustrate this.

Consider the equation,

$$f(t) = \int_0^t \tfrac{1}{2} f(s)ds + 1,$$

where we see that the kernel, k(t,s), is $\frac{1}{2}$, and the forcing function is 1, and so we follow the Neumann series as follows.

$$\begin{aligned} f_1(t) &= 1. \\ f_2(t) &= g + \boldsymbol{k}g, \\ &= 1 + \int_0^t \tfrac{1}{2} ds, \\ &= 1 + \tfrac{1}{2}t. \\ f_3(t) &= 1 + \tfrac{1}{2}\int_0^t (1 + \tfrac{1}{2}s)ds, \\ &= 1 + \tfrac{1}{2}\left[s + \tfrac{s^2}{4}\right]_0^t, \\ &= 1 + \tfrac{t}{2} + \tfrac{t^2}{8}. \\ f_4(t) &= 1 + \tfrac{1}{2}\int_0^t 1 + \tfrac{s}{2} + \tfrac{s^2}{8} ds, \\ &= 1 + \tfrac{t}{2} + \tfrac{t^2}{8} + \tfrac{t^3}{48}. \end{aligned}$$

Extrapolating from this result we observe that the sequence is converging toward the function $e^{\frac{1}{2}}$. The Neumann series is a good solution method for only the simplest of equations, otherwise large, often complex results are obtained. In such cases the use of a mathematical software tool is advantageous, as we shall see later, when such a tool is applied.

Applying the Neumann series to Fredholm integral equations of the second kind is the same as with Volterra equations, except for the integral bounds.

5.2 Numerical methods for linear Volterra equations of the second kind.

When applying a numerical method to an integral equation of Volterra type, the nature of the bounds on the integral dictate that a successive approximation approach be used to obtain a solution. Equations of Fredholm type will require a number of simultaneous equations which need to be solved to find the solution.

If we take our initial formula for a linear Volterra equation of the second kind, and express it as follows,

$$f(nh) = g(nh) + \lambda \int_0^{nh} K(nh, s) f(s) ds, \qquad (5.1)$$

where $t=nh$, and n is any positive integer, we can derive a general formula for the numerical approximation of the integral, thus,

$$\int_0^{nh} \varphi(s) ds \cong \sum_{j=0}^{n} w_{nj} \varphi(jh), \qquad (5.2)$$

where w_{nj} are the given weights.

The values of the weights are dependant upon the type of approximation rule to be used.

5.2.1 The Trapezium rule method.

One of the most elementary numerical solution methods is the trapezoidal method. When applied to the equation (2.1), it has the form,

$$f(nh) = g(nh) + h(\tfrac{1}{2}K(nh,0)\tilde{f}(0) + \sum_{j=1}^{n-1} K(nh,jh)\tilde{f}(jh) + \tfrac{1}{2}K(nh,nh)\tilde{f}(nh)).$$

The accuracy of the trapezium rule is said to have an observed error approximately equal to the step-size squared, i.e. $O(h^2)$ (Baker [4] pp. 761-762). The next two examples, Simpson's 3/8ths and Gregory's method, have a greater order of accuracy: $O(h^5)$ and $O(h^4)$, respectively.

5.2.2 Simpson's 3/8ths method.

The repeated Simpson's rule requires the implementation of the 3/8ths rule for values of n which are odd. When the 3/8ths rule is applied to the last four points in the function, the accuracy of the solution is far better over large intervals, than if applied to the first points in the function (Delves and Walsh [13] pp. 155). For values of n which are less than five, the approximation formula's used for $\tilde{f}(h), \tilde{f}(2h), \tilde{f}(3h), \tilde{f}(4h)$, are taken as,

$$\tilde{f}(2h) = g(2h) + h\{\tfrac{1}{3}K(2h,0)\tilde{f}(0) + \tfrac{4}{3}K(2h,h)\tilde{f}(h) + \tfrac{1}{3}K(2h,2h)\tilde{f}(2h)\}.$$

$$\tilde{f}(3h) = g(3h) + h\{\tfrac{3}{8}K(3h,0)\tilde{f}(0) + \tfrac{9}{8}K(3h,h)\tilde{f}(h) + \tfrac{9}{8}K(3h,2h)\tilde{f}(2h) + \tfrac{3}{8}K(3h,3h)\tilde{f}(3h)\}.$$

$$\begin{aligned}\tilde{f}(4h) = g(4h) + h\{&\tfrac{1}{3}K(4h,0)\tilde{f}(0) + \tfrac{2}{3}K(4h,h)\tilde{f}(h) + \tfrac{4}{3}K(4h,2h)\tilde{f}(2h)\\ &+ \tfrac{2}{3}K(4h,3h)\tilde{f}(3h) + \tfrac{1}{3}K(4h,4h)\tilde{f}(4h)\}.\end{aligned}$$

where $\tilde{f}(h)$ is obtained from the trapezium rule. Since the trapezium rule has an observed error of $O(h^2)$, and Simpson's rule is $O(h^5)$, the value of h used for the trapezium rule approximation is set to $\sqrt{h^5}$.

For values of $n \geq 5$ the following weights are used in equation (5.2).

For even n,

$$w_0^n = \tfrac{1}{3};\ w_{2i}^n = \tfrac{2}{3};\ w_{2i+1}^n = \tfrac{4}{3};\ w_n^n = \tfrac{1}{3}, \qquad i = 0,1,2,\ldots,(n/2)-1.$$

For odd n,

$$\begin{aligned}w_0^n &= \tfrac{1}{3},\\ w_{2i}^n &= \tfrac{2}{3},\quad i = 1,2,3,\ldots,(n-5)/2,\\ w_{2i+1}^n &= \tfrac{4}{3},\quad i = 0,1,2,\ldots,(n-5)/2,\\ w_{n-3}^n &= \tfrac{17}{24},\\ w_{n-2}^n &= w_{n-1}^n = \tfrac{9}{8},\\ w_n^n &= \tfrac{3}{8}.\end{aligned}$$

These weights can then be substituted into the numerical approximation of the integral (5.2), to obtain further approximations.

5.2.3 The Gregory quadrature method.

Developed in the mid-seventeenth century by the Scottish mathematician James Gregory, this method is one of the more stable and accurate methods of approximation when applied to linear Volterra equations of the second kind (Baker [4] pp. 781-784). Simple numerical approximation methods, such as the trapezoidal method, require very small step sizes in their computations to achieve reasonably accurate solutions; but to obtain a more accurate solution, a more accurate integration method must be used.

The weights for this formula are determined by the order of integration to be used, which in our case will be fourth order integration. As a general rule, a large step size will require a large order of integration to ensure a reasonable degree of accuracy. Before computational aids this was often the ideal, because using very small step sizes would require a lot of calculating time.

The q^{th} Gregory quadrature rule is given by,

$$\int_0^{nh}\varphi(s)ds = \sum_{j=0}^{n}{}''\varphi(jh) - h\sum_{i=1}^{q} c_i\left(\nabla^i\varphi(nh) + (-1)^i\Delta^i\varphi(0)\right), \tag{5.3}$$

where c_i is given by,

$$c_i = \sum_{j=1}^{p}(-1)^{i-2j+1}\frac{B_{2j}S_i^{(2j-1)}}{(2j)!\prod_{\psi=2j}^{i}\psi}, \qquad \text{where } p=\left\lfloor\tfrac{i+1}{2}\right\rfloor.$$

The terms B_{2j} and $S_i^{(j)}$ are given by Bernoulli and Sterling number equations (Brunner and van der Houwen [7], pp. 73-74). The first four coefficients are, $c_1 = \frac{1}{12}$, $c_2 = \frac{1}{24}$, $c_3 = \frac{19}{720}$, $c_4 = \frac{3}{160.}$ At this point it is imperative to note that the q^{th} Gregory rule has the order of $q+2$, and we will therefore be using the second Gregory rule for our fourth order Gregory method.

Observing the Gregory formula (5.3), we can see that it is simply just the trapezium rule with correcting values. These values however, produce one of the most accurate quadrature methods for second order linear equations of Volterra type. The smallest value of n which can be used is 5, leaving four required starting values, f(h), f(2h), f(3h), and f(4h), where f(0)=g(0). Using q=2 in equation (5.3), we obtain.

$$\begin{aligned}\int_0^{nh}\varphi(s)ds &= h(\tfrac{1}{2}\varphi(0)+\varphi(h)+\varphi(2h)+\ldots+\varphi(nh-h)+\tfrac{1}{2}\varphi(nh))\\ &\quad -\tfrac{h}{12}(\nabla\varphi(nh)-\Delta\varphi(0))-\tfrac{h}{24}(\nabla^2\varphi(nh)+\Delta^2\varphi(0)).\end{aligned}$$

$$\begin{aligned}&= h\left(\tfrac{1}{2}\varphi(0)+\varphi(h)+\varphi(2h)+\ldots+\varphi(nh-2h)+\varphi(nh-h)+\tfrac{1}{2}\varphi(nh)\right)\\ &\quad -\tfrac{h}{12}\left(\varphi(nh)-\varphi(nh-h)-\varphi(h)+\varphi(0)\right)\\ &-\tfrac{h}{24}\left(\varphi(nh)-2\varphi(nh-h)+\varphi(nh-2h)+\varphi(2h)+\varphi(2h)-2\varphi(h)+\varphi(0)\right).\end{aligned}$$

which eventually becomes,

$$\int_0^{nh}\varphi(s)ds = h(\tfrac{3}{8}\varphi(0)+\tfrac{7}{6}\varphi(h)+\tfrac{23}{24}\varphi(2h)+\varphi(3h)+\ldots$$
$$\ldots+\varphi(nh-3h)+\tfrac{23}{24}\varphi(nh-2h)+\tfrac{7}{6}\varphi(nh-h)+\tfrac{3}{8}\varphi(nh)).$$

The implementation of such a quadrature method will require a set of five starting values. The initial approximation is always taken as $\tilde{f}(0)=g(0)$. If the exact solution is known then this can be used. However, in this thesis we shall be using the trapezium rule to generate $\tilde{f}(h)$, and Simpson's rule to generate $\tilde{f}(2h), \tilde{f}(3h), \tilde{f}(4h)$. Gregory's rule will then be used to complete the approximation. Due to the trapezium rule being of order two whereas, both Simpson's rule and the Gregory rule are of order five and four respectively, the stepsize used in the trapezium rule will need to be chosen so the errors are in line with those obtained via the fourth order methods. Below are the equations to be used in finding the starting values.

The Trapezium integration approximation.

$$\int_0^{nh}\varphi(s)ds = h(\tfrac{1}{2}K(nh,0)\tilde{f}(0)+\sum_{j=1}^{n-1}K(nh,jh)\tilde{f}(jh)+\tfrac{1}{2}K(nh,nh)\tilde{f}(nh)).$$

$$\therefore \quad \tilde{f}(h)=g(h)+h\{\tfrac{1}{2}K(h,0)\tilde{f}(0)+\tfrac{1}{2}K(h,h)\tilde{f}(h)\}.$$

where $h=h^2$.

Simpson's integration approximation.

Simpson's rule (repeated Simpson's rule) is used to obtain $\tilde{f}(2h), \tilde{f}(4h)$, and Simpson's ⅜ rule is used for $\tilde{f}(3h)$ (the reason for this being that Simpson's rule is only valid for n even). Thus our remaining starting values are given by ,

$$\tilde{f}(2h)=g(2h)+h\{\tfrac{1}{3}K(2h,0)\tilde{f}(0)+\tfrac{4}{3}K(2h,h)\tilde{f}(h)+\tfrac{1}{3}K(2h,2h)\tilde{f}(2h)\}.$$

$$\tilde{f}(3h)=g(3h)+h\{\tfrac{3}{8}K(3h,0)\tilde{f}(0)+\tfrac{9}{8}K(3h,h)\tilde{f}(h)+\tfrac{9}{8}K(3h,2h)\tilde{f}(2h)+\tfrac{3}{8}K(3h,3h)\tilde{f}(3h)\}.$$

$$\tilde{f}(4h)=g(4h)+h\{\tfrac{1}{3}K(4h,0)\tilde{f}(0)+\tfrac{2}{3}K(4h,h)\tilde{f}(h)+\tfrac{4}{3}K(4h,2h)\tilde{f}(2h)$$
$$+\tfrac{2}{3}K(4h,3h)\tilde{f}(3h)+\tfrac{1}{3}K(4h,4h)\tilde{f}(4h)\}.$$

As well as having algorithms whereby the starting values are determined in this manner, there will be routines whose starting values can be entered by the user (see appendix one).

5.3.1 A simple numerical method for linear Volterra equations of the first kind.

The ill-posed nature of first kind equations create problems when the above mentioned numerical methods are applied. It could be argued that Gregory's method and Simpson's 3/8ths method are very inaccurate when attempting to approximate a first kind equation in it's discretised form. Even though the methods themselves are convergent, when they are applied to linear Volterra equations of the first kind, they become non convergent (Linz [15], p. 143). As described in chapter four, §4.1.2, some first kind equations may be analogous to equations of the second kind and

consequently solved. To cover a wide range of solution techniques for integral equations a numerical method which is applicable to first kind equations that cannot be solved by methods described in chapter four, would be preferable. Rejecting higher order numerical methods leaves methods with low accuracy such as the trapezoidal method, Euler's method and the mid-point method. The latter of these methods is the chosen one for linear Volterra equations of the first kind as we shall discover in the ensuing discussion.

We will begin with the standard linear Volterra equation of the first kind,

$$g(t) = \int_0^t K(t,s)f(s)ds, \quad 0 \le t \le T. \tag{5.4}$$

and approximate the integral by,

$$\int_0^{t_n} u(s)ds = h\sum_{i=0}^{n-1} u(t_n, t_{i+1/2})\tilde{f}_{i+1/2}.$$

This is the mid-point rule formula and has an accuracy order of $O(h^2)$. Since it can be shown that the difference between the exact solution and the approximation is

$$\tilde{f}_{n+1/2} = f(t_{n+1/2}) + \frac{h^2}{24}e(t_{n+1/2}) + O(h^4),$$

we can apply Richardson's extrapolation method to improve accuracy. This involves computing the solution with two different stepsizes and then subtracting the two results, thus eliminating the error term. So, let

$$R(t,h) = f(t) + \frac{h^2}{24}e(t) + O(h^4), \tag{5.5}$$

and,

$$R\left(t, \frac{h}{3}\right) = f(t) + \frac{h^2}{216}e(t) + O(h^4). \tag{5.6}$$

To eliminate the error term we must multiply (5.6) by nine and then subtract from (5.5), to leave us with,

$$f(t) = \frac{9}{8}R\left(t, \frac{h}{3}\right) - \frac{1}{8}R(t,h) + O(h^4).$$

When this method is applied to the mid point rule the approximation will have an observed error of $O(h^4)$. Since we may extrapolate the mid-point rule, and because it is stable in comparison to other numerical approximations when applied to first kind equations, the mid-point rule would appear to be the favoured choice (Linz [15] p147).

5.3.2 Numerical solution method for Abel type equations.

The complexity of Abel type equations creates a lack of suitable analytical solution methods. Analytical methods which do exist are extremely limited in the scope of equations to which they can be applied. Producing a piece of software which would only be applicable to a very limited number of problems seems purposeless. For this reason, I have chosen to present only a numerical solution method for linear Volterra equations of Abel type. Also, because the scope for applying Abel equations is so confined, the background theorems on the topic have been excluded. These theorems can be seen in Linz [15] pp. 71-75. Only a numerical method will briefly be described.

Therefore, taking a linear Volterra equation of Abel type,

$$\int_0^t \frac{K(t,s)}{(t-s)^u} f(s)ds = g(t), \tag{5.6}$$

and applying the following restrictions,

- K(t,s) is smooth,
- $K(t,t) \neq 0$,
- $0<u<1$,

we can apply the mid point method and using product integration approximate the integral of (5.6) as,

$$\int_0^t (t-s)^{-u} K(t,s) f(s)ds = \sum_{i=0}^{n-1} K(t_n, t_{i+1/2}) \tilde{f}_{i+1/2} \int_{t_i}^{t_{i+1}} (t-s)^{-u} ds.$$

The ill-posed nature of Abel equations and the singularity in the kernel make the mid-point method somewhat inaccurate (Linz [15], pp. 165-168), but Richardson's extrapolation method may again be applied in conjunction with Aitken's error correction term to improve the accuracy. Hence, the extrapolated value is produced by,

$$f(t) = R\left(t, \frac{h}{9}\right) - \frac{1}{3^q - 1}\left(R\left(t, \frac{h}{9}\right) - R\left(t, \frac{h}{3}\right)\right),$$

where q is given by,

$$q = \log_3\left(\frac{R(t,h/3) - R(t,h)}{R(t,h/9) - R(t,h/3)}\right).$$

5.4 Numerical methods for linear Fredholm equations of the second kind.

The method of finding the unknowns for Fredholm equations is different to that of Volterra equations in that a series of simultaneous equations are produced and then solved. As an example, we shall consider the trapezium rule applied to the following equation.

$$f(t) = g(t) + \alpha \int_a^b K(t,s) f(s)ds, \tag{5.7}$$

The numerical approximation of the integral becomes,

$$\int_a^b u(s)ds = \tfrac{h}{2}\{u(a) + 2\sum_{j=1}^{n-1} u(a+jh) + u(b)\}, \tag{5.8}$$

where n=(b-a)/h.

Applying (5.8) to (5.7), and letting n=4 gives

$$f(t) = g(t) + \tfrac{\alpha h}{2}\{K(t,a)\tilde{f}(a) + 2K(t,a+h)\tilde{f}(a+h) + 2K(t,a+2h)\tilde{f}(a+2h) + 2K(t,a+3h)\tilde{f}(a+3h) + K(t,b)\tilde{f}(b)\}.$$

Now to obtain the values of $\tilde{f}(a), \tilde{f}(a+h), \tilde{f}(a+2h), \tilde{f}(a+3h), \tilde{f}(b)$, we substitute values of t=a, t=a+h, t=a+2h, t=a+3h, t=b, into the above equation, producing five simultaneous equations which can then be solved.

This technique may be applied to Simpson's 3/8ths and Gregory's method using the weights described previously, noting that obviously no starting values will be required as compared with Volterra equations.

Chapter Six

Concluding remarks.

In this conclusion, I will be primarily concerned with describing the achievements that have been made in this thesis and how they compare to the objectives outlined in the introduction. This will include a summary discussing the established mathematical methods and how they have been used to produce the new software routines presented in the appendices. Finally, I will be presenting a list of further work that could be done and how the software packages could be further developed.

What has been achieved?

The aim of this thesis was to take existing mathematical solution methods for integral equations (the Laplace transform technique, for example) and produce new and original software algorithms that can solve a variety of linear integral equations, of both Volterra and Fredholm type. I have succeeded in creating two software packages (one for Fredholm integral equations and one for Volterra integral equations) that can be used in conjunction with the Maple mathematical software tool to produce solutions to a large variety of integral equations. These new software packages contain analytical and numerical solution techniques that are designed to be easy to use and familiar to anyone who has used Maple (help pages have also been created for every routine). The types of integral equations that the new and original algorithms can solve, are listed below, along with the methods used.

1. Linear Volterra integral equations: Analytical solution algorithms.

i). Equations of the first kind with convolution kernels can be solved using the Laplace transform solution method.
ii). General equations of the first kind can be solved using the method of conversion to an ordinary differential form.
iii). Equations of the second kind with convolution kernels can be solved using Laplace transforms.
iv). Equations of the second kind of a more general form can be solved by a conversion to an ordinary differential form.
v). The resolvent kernel solution method can be used on equations of the second kind with a complex forcing function and a convolution kernel.

Numerical solution algorithms.

vi). The Neumann iterative series can be applied to equations of the second kind.
vii). The Trapezium rule, Simpson's 3/8ths rule and Gregory's rule can all be applied to equations of the second kind, with or without starting values (where applicable).
viii). Equations of the first kind can be solved using the mid point rule with Richardson's extrapolation.
ix). Abel type equations can be solved using the mid point rule with Richardson's extrapolation and Aitken's error correction terms.

2. Linear Fredholm integral equations: Analytical solution algorithms.

i). Equations of the second kind can be solved by conversion to an ordinary differential form.

Numerical solution algorithms.

ii). The Neumann iterative series can be applied to equations of the second kind.
iii). The Trapezium rule, Simpson's rule and Gregory's rule can all be applied to equations of the second kind.

The emphasis for the solution algorithms leans more towards Volterra integral equations because there are a few programs to be found on the Internet for the solution of Fredholm integral equations. The solution algorithms listed here for Fredholm equations are, however, unique. Volterra solution algorithms are extremely rare and so the majority of the work is concerned within this area.

Although equations of the forms listed above can be applied to the software routines, many problems do not have solutions. The algorithms have been coded to detect such problems and equations without solutions or equations with singularities will be found and the user informed.

Further work.

There are several areas where the software packages could be developed further.

1. An extension to the types of integral equations in the packages. Forms of integral equations which could be catered for include:
   a) Integro-differential equations. The solution routine for this kind of equation could be accomplished by making slight amendments to the *code* routine
   b) Higher order integral equations. Since equations of this type can be considered as a system of first order equations, existing routines could be used and modified.
   c) Non linear equations.

2. Develop two routines which could check for the existence and uniqueness of a solution to a problem. These routines would be based on:
   a) An algorithm that could apply the conditions of Picard's iterative theorem to Volterra integral equations. This would involve checking that the kernel and forcing function are continuous over the interval of the integral, although more work would be required to determine if this is possible.

b) An algorithm that could apply the conditions of the Fredholm alternative theorem to Fredholm integral equations.

3. The addition of sub routines to the numerical solution methods in an attempt to analyse the stability of the results.

Appendix One.

Volterra Maple package description.

The following section provides a documentation of the routines available in the Volterra package. To run a routine the command
> with(volterra);
must be entered, after which any of the following can be used;

- **code** - produces a solution to a linear Volterra integral equation of the second kind by means of a conversion to its ordinary differential form.
- **codeone** - as above but for a linear Volterra integral equation of the first kind.
- **voltlap** - Uses Laplace transforms to obtain a solution to a linear Volterra integral equation of the second kind with a convolution kernel.
- **voltone** - as above but for an equation of the first kind.
- **voltres** - produces a solution to a linear Volterra integral equation of the second kind with a complex kernel, and by the application of Laplace transforms, expresses the solution in resolvent kernel form.
- **volttrap** - trapezium numerical solution method for linear Volterra integral equations of the second kind.
- **voltsimp** - as above but using Simpson's 3/8ths method.
- **voltgreg** - as above but using Gregory's method.
- **simpini** - Simpson's numerical approximation method using initial values.
- **gregory** - Gregory's numerical approximation method using initial values.
- **voltneum** - Neumann series approximation to a linear Volterra integral equation of the second kind.
- **firstmid** - mid point numerical approximation for a linear Volterra integral equation of the first kind with an option for Richardson's extrapolation.
- **Abelmid** - mid point numerical approximation for a linear Volterra integral equation of the first kind of Abel type with an option for Richardson's extrapolation containing Aitken's correction terms.

Each routine contains its own help page (within Maple), and can be accessed by typing the following command,

> *?progman_name*;

An example of a help page can be seen in appendix three.

'code' - Maple routine.

1. Purpose.

code produces a solution to a linear Volterra equation of the second kind by means of a conversion to its ordinary differential form.

2. Specification.

```
with(volterra):
code(G,K);
```

Algebraic expressions:- G, K.

3. Description.

code will convert a linear Volterra equation of the second kind of the form,

$$f(t) = g(t) + \alpha \int_0^t K(t,s) f(s) ds, \quad 0 \le t \le T.$$

into an ordinary differential equation. It is assumed that the functions involved are sufficiently smooth and well behaved. Some equations containing singularities or an ill-posed problem can be converted to their ordinary differential form, but a solution of this form may not be attainable. The routine will differentiate the equation until an appropriate substitution for the integral term is found. A maximum number of ten derivatives of the original equation will be examined to find a substitution. However, in general only one or two differentiations will be required. If an equivalent ordinary differential equation form is found, the initial values will be computed and stored in an array of real type. The values can then be used in conjunction with Maple's built in *dsolve* routine to find a solution for f(t). The theory for this method can be found in chapter four, section 4.1.1.

4. Parameters.

Input parameters.

G - The forcing function of the equation. If this function contains an independent variable it must be of the form *t*. *G* must be of algebraic type, as defined and recognised by Maple.

K - The kernel of the equation. The independent variables of the equation must be of the form *s* and *t*. *K* must be of algebraic type, as recognised by Maple.

Output parameters.

ODE1 - The entered equation.

ODE - The equivalent ordinary differential equation. This variable is defined as local so it may be used in the *dsolve* routine.

const - Once an ordinary differential equation has been found, the initial values will be computed and stored in a real array. These may be accessed by typing *print(const);* and used with *dsolve* to find a solution for f(t).

5. Error messages.

ERROR, code expects its first/second argument to be of type algebraic.

The forcing function *G* and the kernel *K* must be of algebraic type. If not then this error message is encountered.

ERROR, there is no equivalent ODE up to tenth order.

The routine could not find a suitable substitution of the integral term and so an equivalent ordinary differential equation could not be found. Note that a kernel of the form t^n, where n>9 will have an equivalent ordinary differential equation, but since the routine performs a maximum number of ten differentiations, such a function will not be recognised.

ERROR, (in code) division by zero.

This error message occurs when the routine is establishing the initial values for the ODE. The routine finds values for f(0), the derivative of f(0) and so on. If, for example, the forcing function *G* is 1/t, such an error message will occur. A singularity in the equation or resultant computations is usually the main cause for this error message.

NULL.

If NULL is produced when using *dsolve* and the initial values there is no solution for f(t) of the equivalent ordinary differential equation.

6. Accuracy.

There are no accuracy measurements.

7. Further comments.

If a solution is required for f(t) the routine is best suited to well behaved problems. The routine may well find an equivalent ordinary equation of an ill-posed problem, but a solution for f(t) will be unlikely. Only a limited number of equations do have an equivalent differential form. In particular, convolution equations are well suited to this solution method. Trigonometric functions are also ideal for this method because they readily change into a form which can be substituted upon differentiation

8. Examples.

Example 1. Solve the linear convolution equation,

$$f(t) = t + \tfrac{1}{2}\int_0^t \cos(t-s)f(s)ds.$$

Listing of inputs and outputs for execution of problem.

```
> with(volterra):

> code(t,1/2*cos(t-s));
```

Equation entered: $f(t) = t + \frac{1}{2}\int_0^t \cos(t-s)f(s)ds.$

Equivalent ordinary differential equation: $\frac{\partial^2}{\partial t^2} f(t) = -f(t) + t + \frac{1}{2}\left(\frac{\partial}{\partial t} f(t)\right).$

```
> print(const);
```

```
table([
1=0
2=1
])
```

```
> dsolve({ODE,f(0)=0,D(f)(0)=1},f(t));
```

$$f(t) = t + \frac{1}{2} + \frac{1}{30}\sqrt{3}\sqrt{5}e^{\frac{1}{4}t}\sin\left(\frac{1}{4}\sqrt{3}\sqrt{5}t\right) - \frac{1}{2}e^{\frac{1}{4}t}\cos\left(\frac{1}{4}\sqrt{3}\sqrt{5}t\right).$$

Example 2.

We shall now examine another convolution equation, but with a more complex kernel. Solve the equation,

$$f(t)=\frac{1}{2}t^2+\frac{1}{\alpha}\int_0^t\left(\alpha\sin(t-s)+\cos(t-s)\right)f(s)ds.$$

Listing of inputs and outputs for execution of problem.

```
> with(volterra):

> code((1/2)*t^2,(sin(t-s)+1/alpha*cos(t-s)));
```

Equation entered: , $f(t)=\frac{1}{2}t^2+\frac{1}{\alpha}\int_0^t\left(\alpha\sin(t-s)+\cos(t-s)\right)f(s)ds.$

Equivalent ordinary differential equation: ,

$$\frac{\partial^4}{\partial t^4}f(t)=-\frac{1}{2}t^2-\frac{1}{\alpha}\left(\frac{\partial}{\partial t}f(t)\right)+\left(\frac{\partial^2}{\partial t^2}f(t)\right)+\frac{1}{\alpha}\left(\frac{\partial^3}{\partial t^3}f(t)\right)..$$

```
> print(const);
```

```
table([
1=0
2=0
3=1
4=1/α
])
```

```
> dsolve({ODE,f(0)=0,D(f)(0)=0,(D@@2)(f)(0)=1,(D@@3)(f)(0)=1/α},f(t));
```

$$f(t)=-\alpha^4-t\alpha^3-\tfrac{1}{2}t^2\alpha^2-\alpha^2-\tfrac{1}{6}t^3\alpha-t\alpha+\left(\alpha^4+\alpha^2\right)e^{\frac{t}{\alpha}}$$

Example 3.

This next example is a perfect illustration of how a linear Volterra equation of the second kind can be converted into an ordinary differential equation which has no solution. Solve,

$$f(t)-\int_0^t\sin(t)f(s)ds=\frac{1}{t+1}+\sqrt{(t+1)}.$$

Listings of inputs and outputs.

```
> with(volterra):

> code(1/(t+1)+sqrt(t+1),sin(t));
```

Equation entered: , $f(t)=\frac{1}{t+1}+\sqrt{t+1}+\int_0^t\sin(t)f(s)ds.$

Equivalent ordinary differential equation: ,

$$\frac{\partial^2}{\partial t^2}f(t)=2\frac{1}{(t+1)^3}-\frac{1}{4}\frac{1}{(t+1)^{3/2}}-f(t)+\frac{1}{t+1}+\sqrt{t+1}+2\cos(t)f(t)+\sin(t)\left(\frac{\partial}{\partial t}f(t)\right)..$$

```
> print(const);
```

```
table([
1=2
2=-1/2
])
```

```
> dsolve({ODE,f(0)=2,D(f)(0)=-1/2},f(t));
```

Note. Although this problem yields an equivalent ordinary differential equation, there is no solution for f(t) when using the *dsolve* routine.

9 Program listing.

#Conversion to ODE form.

```
restart;
code:=proc(G:algebraic,K:algebraic)
local ODE1,standin,Intsub,count,EndIntsub,Gdsolve;
const:=array[1..10];
const[1]:=eval(subs(t=0,G));
ODE1:=f(t)=G+Int(K*f(s),s=0..t);
standin:=value(subs(t=v,G+Int(K*f(s),s=0..v)));
f(v):=standin;
Intsub:=Int(K*f(s),s=0..t);
ODE:=diff(ODE1,t);
count:=1;
#Loop to establish whether differential equations contain
#the original integral.
while has(integrand(ODE),integrand(Intsub))=false and
has(integrand(ODE),-integrand(Intsub))=false and  count<10 do
if not integrand(ODE)={} then
if type(integrand(ODE)/integrand(Intsub),constant)=true then
break;
fi;
fi;
if integrand(ODE)={} then
count:=11 else
count:=count+1;
if count=10 then
ERROR(`There is no equivalent ODE up to tenth order.`);
fi;
#Find initial values for dsolve use.
Gdsolve:=value(subs(t=v,rhs(ODE)));
const[count]:=value(subs(v=0,Gdsolve));
ODE:=diff(ODE,t);
fi;
od;
if count<10 then
print(`Equation entered:  `,ODE1);
#Find substitution.
if not (integrand(Intsub)={} or integrand(ODE)={})then
EndIntsub:=(integrand(ODE)/integrand(Intsub))*(f(t)-G);
else EndIntsub:=f(t)-G;
fi;
ODE:=subs(Int(integrand(ODE),s=0..t)=EndIntsub,ODE);
print(`Equivalent ordinary differential equation:  `,ODE);
```

```
elif count=11 then print(`Equation entered:  `,ODE1);
print(`Equivalent ordinary differential equation:  `,ODE);
else lprint(`There is no equivalent ordinary differential equation, (up to 10th order).`);
fi;
end;
```

10 Programming problems and effectiveness of Maple.

A principle feature of the *code* routine are algorithms that can identify when a substitution for the integral can be made. Two separate checks are required to establish whether a substitution is possible. The code for these being,

- has(integrand(ODE),integrand(Intsub)).

This statement will check the differentiated integrand with the original integrand, and if the former contained the latter the substitution will be made. However, if for example the original integrand is negative and the differentiated integrand is positive, Maple will not recognise the two as being similar. This is not true though for a negative differentiated integrand when compared to a positive original integrand.

Example.

Taking example 1 from before, Maple will recognise

$$-\int_0^t \cos(t-s)f(s)ds$$

as being alike to an original integral of

$$\int_0^t \cos(t-s)f(s)ds,$$

so the substitution would be f(t)-g(t), but will not recognise the integral

$$\int_0^t \cos(t-s)f(s)ds$$

as being alike to an original integral of

$$-\int_0^t \cos(t-s)f(s)ds,$$

and so the following line of code was needed.

- has(integrand(ODE),-integrand(Intsub))

The second check required deals with the constants of the integrands. The problem here occurs when the constant of the original integrand is dissimilar to that of the differentiated integrand. For example, the differentiated integrand $4\sin(t-s)$ would not be recognised as being alike to an original integrand of $\frac{1}{2}\sin(t-s)$. To overcome this problem a check was incorporated where the differentiated integrand is divided by the original integrand and if the result is a constant the routine would make a substitution.

'codeone' - Maple routine.

1. Purpose.

codeone produces a solution to a linear Volterra equation of the first kind by means of a conversion to its ordinary differential form.

2. Specification.

```
with(volterra):
codeone(G,K);
```

Algebraic expressions:- G, K.

3. Description.

codeone will convert a linear Volterra equation of the first kind with a smooth kernel of the form,

$$g(t) = \int_0^t K(t,s)f(s)ds, \quad 0 \le t \le T.$$

into an ordinary differential equation or a solution for f(t). Depending on the form of the kernel the routine will display a solution for f(t) or a solution containing derivatives of f(t). If the latter is true the initial values can be obtained by typing *print(const);* and then using *dsolve* to find a solution for f(t). It is assumed the functions involved are sufficiently smooth and well behaved. Some equations can be converted to their ordinary differential form, but a subsequent solution may not exist. The routine will differentiate the equation until an appropriate substitution for the integral term is found. A maximum number of ten derivatives of the original equation will be examined to find a substitution. Essentially this is the same practice as used in the *code* routine, except a solution may only be found if the equation produces an f(t) function upon differentiation (see chapter 4, §4.1.2). If an equivalent ordinary differential equation form is found, the initial values will be computed and stored in an array of real type. The values can then be used in conjunction with Maple's built-in *dsolve* routine to find a solution for f(t). The theory behind this solution technique can be found in chapter four, section 4.1.2.

4. Parameters.

Input parameters.

G - The forcing function of the equation. If this function contains an independent variable it must be of the form *t*. *G* must be of algebraic type, as defined and recognised by Maple.

K - The kernel of the equation. The independent variables of the equation must be of the form *s* and *t*. *K* must be of algebraic type, as recognised by Maple.

Output parameters.

ODE1 - The entered equation.

ODE - The equivalent ordinary differential equation or a solution for f(t). This variable is defined as local so it can be used in the *dsolve* routine.

const - Once an ordinary differential equation has been found, the initial values will be computed and stored in a real array. These may be accessed by typing *print(const);* and used with *dsolve* to find a solution for f(t).

5. Error messages.

ERROR, Since g(0) is not equal to zero, a solution does not exist.

For a solution to exist the forcing function must equal zero at t=0.

ERROR, codeone expects its first/second argument to be of type algebraic.

The forcing function G and the kernel K must be of algebraic type. If not then this error message is encountered.

ERROR, Singularity encountered. f(t) does not exist at t=0.

The routine found an expression for f(t), but when t=0 the expression was found to be erroneous- Most often due to a singularity.

ERROR, there is no equivalent ODE up to tenth order.

The routine could not find a suitable substitution of the integral term and so an equivalent ordinary differential equation could not be found. Note that a kernel of the form t^n, where n>9, will have an equivalent ordinary differential equation; but since the routine performs a maximum number of ten differentiations, such a function will not be recognised.

Maple could not verify the solution to the equation.

Part of the solution checking algorithm. See section 6 below.

The solution has been checked and found to be erroneous.

The solution verification algorithm has checked the computed solution and found it to be incorrect. This is only applicable when the solution is explicit in nature and does not contain derivatives. See section 6 below.

NULL.

If NULL is produced when using *dsolve* and the initial values, there is no solution for f(t) of the equivalent ordinary differential equation.

6. Accuracy.

The routine contains a solution verification algorithm and if an explicit solution is given (one not containing derivatives needing *dsolve* to find a solution for f(t)) it will check the solution in the integral and compare it with the forcing function. An example of a problem which yields a solution that is found to be incorrect can be seen in section 8, example 3. The checker is based on numerical substitution because analytical comparisons in Maple are complex (see section 10).

7. Further comments.

If a solution is required for f(t) then the routine is best suited to well behaved problems. The routine can find an equivalent ordinary equation for an ill-posed problem, but a solution for f(t) will be unlikely. Only a limited number of equations do have an equivalent differential form. In particular, trigonometric functions and convolution equations are well suited to this solution method.

8. Examples.

Example 1. This first example shows how a simple linear Volterra equation of the first kind can be solved. Solve,

$$\frac{1}{2}t^2 = \int_0^t (t-s)f(s)ds.$$

Listing of inputs and outputs for execution of problem.

> with(volterra):

> codeone(1/2*t^2,t-s);

$$\textit{Equation entered: } , \frac{1}{2}t^2 = \int_0^t (t-s)f(s)ds.$$

$$\textit{Solution of equation: } , f(t) = 1.$$

This example shows how a first kind equation can yield an explicit solution. In the next example we will show an equation which does not yield an explicit solution, but is shown to have an equivalent ordinary differential form.

Example 2.

Solve the equation,

$$\int_0^t \cos(t)f(s)ds = \frac{1}{\sqrt{(t+1)^3}} - 1.$$

Listing of inputs and outputs for execution of problem.

> with(volterra):

> codeone(1/(t+1)^(-3/2),cos(t));

$$\textit{Equation entered: } , \frac{1}{\sqrt{(t+1)^3}} - 1 = \int_0^t \cos(t)f(s)ds.$$

Equivalent ordinary differential equation:

$$\frac{15}{4}\frac{1}{(t+1)^{7/2}} + \frac{1}{(t+1)^{3/2}} - 1 + 2\sin(t)f(t) - \cos(t)\left(\frac{\partial}{\partial t}f(t)\right) = 0.$$

> print(const);

```
table([
1=-3/2
])
```

> dsolve({ODE,f(0)=-3/2},f(t));

$$f(t) = -\frac{3}{2}\frac{1}{\cos(t)(t+1)^{5/2}} + \frac{\sin(t)t}{\cos(t)^2(t+1)^{5/2}} + \frac{\sin(t)}{\cos(t)^2(t+1)^{5/2}}$$
$$-\frac{\sin(t)t^2}{\cos(t)^2(t+1)^2} - 2\frac{\sin(t)t}{\cos(t)^2(t+1)^2} - \frac{\sin(t)}{\cos(t)^2(t+1)^2}.$$

This is a good example of how the routine will determine whether the equation is a solution for f(t) or an ordinary differential equation. The example is of the latter form, where once the routine determines the form of the equation it will begin to calculate the initial values.

Example 3.

This problem produces a solution which does not satisfy the original equation. It is a good example of the ill-posed nature of linear Volterra integral equations of the first kind where the solution contains the Dirac function.

Solve,

$$\frac{1}{2}e^t - \sin(t) - \cos(t) + \frac{1}{2} = \int_0^t \sin(t-s)f(s)ds.$$

Listing of inputs and outputs.

```
> with(volterra):

> codeone(1/2*exp(t)-sin(t)-cos(t)+1/2,sin(t-s));
```

Equation entered: $, \frac{1}{2}e^t - \sin(t) - \cos(t) + \frac{1}{2} = \int_0^t \sin(t-s)f(s)ds.$

Solution of equation: $, f(t) = e^t + \frac{1}{2}$

This solution has been checked and found to be erroneous.

The exact solution to the equation includes the *Dirac* function, so

$$f(t) = -\frac{1}{2}\text{Dirac}(t) + e^t + \frac{1}{2}.$$

9 Program listing.

```
#Conversion to ODE form of 1st order Volterra.

restart;
readlib(isolate);
codeone:=proc(G:algebraic,K:algebraic)
local Solcheck, condition, ODE1, Intsub, count, EndIntsub, InitODE, sing,
standin, Gdsolve, Sol, FuncS, Right;
#Check if solution exists.
SolCheck:=eval(subs(t=0,G));
if not SolCheck=0 then
ERROR(`Since g(0) is not equal to 0, there is no solution.`);
fi;
const:=array[1..10];
condition:=0;
ODE1:=G=Int(K*f(s),s=0..t);
Intsub:=Int(K*f(s),s=0..t);
ODE:=diff(ODE1,t);
count:=1;
#Loop to establish whether differential equations contain
#The original integral.
while has(integrand(ODE),integrand(Intsub))=false and
has(integrand(ODE),-integrand(Intsub))=false and
count<10 do
if integrand(ODE)={} then count:=11 else
count:=count+1;
#Establish initial conditions.
if has(ODE,f(t))=true and condition=0 then
```

```
InitODE:=isolate(ODE,f(t));
sing:=traperror(eval(subs(t=0,rhs(InitODE))));
if sing=`division by zero` then
ERROR(`singularity encountered for f(t), when t=0.`);
fi;
const[1]:=value(subs(t=0,rhs(InitODE)));
standin:=value(subs(t=v,rhs(InitODE)));
f(v):=standin;
condition:=1;
fi;
if condition>0 then
Gdsolve:=value(subs(t=v,rhs(InitODE)));
const[condition]:=value(subs(v=0,Gdsolve));
condition:=condition+1;
fi;
ODE:=diff(ODE,t);
InitODE:=diff(InitODE,t);
fi;
od;
if count<10 or count=11 then
print(`Equation entered:  `,ODE1);
if count<10 then
#Find substitution.
if not (integrand(Intsub))=0 then
EndIntsub:=(integrand(ODE)/integrand(Intsub))*(G);
else EndIntsub:=G;
fi;
ODE:=subs(Int(integrand(ODE),s=0..t)=EndIntsub,ODE);
fi;
if has(ODE,diff(f(t),t)) then
Sol:=isolate(ODE,f(t),1);
else
Sol:=isolate(ODE,f(t));
sing:=traperror(eval(subs(t=0,Sol)));
if sing=`division by zero` then
ERROR(`singularity encountered for f(t), when t=0.`)
fi;
fi;
#Print solution.
print(`Solution of equation:  `,Sol);
#Verify solution.
if not has(Sol,diff(f(t),t)) then
Solsub:=(eval(subs(t=s,rhs(Sol))));
sing:=traperror(value(Int(K*Solsub,s=0..t)));
if sing=lasterror then
print(`Maple could not verify the solution of the equation.`)
else
Right:=value(Int(K*Solsub,s=0..t));
```

```
for i from -2 to 2 do
Num_Right:=evalf(subs(t=i,Right),15);
Num_G:=evalf(subs(t=i,G),15);
Difference:=Num_Right-Num_G;
if (Difference^2)>(1*10^(-10)) then
print(`This solution has been checked and found to be erroneous.`);
break;
fi;
od;
fi;
fi;
else lprint(`There is no solution for f(t), (up to 10th order differential equation).`);
fi;
end;
```

10 Programming problems and effectiveness of Maple.

As the underlying theory of this method is essentially that for linear Volterra equations of the second kind, (chapter four, §§ 4.1.1, 4.1.2), the same features of the *code* routine are required for the *codeone* routine (see *code* documentation §10). Another essential part of the code is used for identifying singularities when a value for f(0) is required. As an example, consider the equation,

$$e^t - t^2 = \frac{1}{\alpha}\int_0^t e^t \sin(s) f(s) ds,$$

whose solution (when using the conversion to ordinary differential form method) is

$$f(t) = -\frac{(2t - t^2)\alpha}{e^t \sin(t)}.$$

Normally Maple will return an *ERROR, division by zero* message when finding a value for f(0), but the routine identifies when singularities will occur.

As mentioned in section 6, a numerical verification algorithm is incorporated into the routine for checking explicit solutions. An analytical verification method would not recognise functions such as,

$$\frac{16}{15}\cos(t)^2 - \frac{8}{15},$$

to be the same as,

$$\frac{16}{15}\cos(2t),$$

without a great deal of manipulation, and thus a numerical solution verification method was chosen here.

'voltlap' - Maple routine.

1. Purpose.

voltlap produces a solution to a linear Volterra integral equation of the second kind of convolution type by the application of Laplace transforms.

2. Specification.

```
with(volterra):
voltlap(G,K,over);
```

Algebraic expressions:- G, K.
optional string:- over.

3. Description.

voltlap will produce a solution to a linear Volterra equation of the second kind of convolution type, of the form,

$$f(t) = g(t) + \int_0^t K(t-s)f(s)ds, \quad 0 \le t \le T.$$

The kernel of the equation must be of convolution type so that the following Laplace transform theorem can be applied.

$$\ell\{K(t)*f(t)\} = \int_0^t K(t-s)f(s)ds.$$

The forcing function *g* must also have a Laplace transform otherwise the equation is best suited to the *voltres* routine, which expresses the solution in resolvent kernel form. The solution method of this routine can be seen in chapter four, §4.3.1.

4. Parameters.

Input parameters.

G - The forcing function of the equation. If this function contains an independent variable it must be of the form *t*. G must be of algebraic type, as defined and recognised by Maple. The function must have a Laplace transform (see above).

K - The kernel of the equation. The independent variables of the equation must be of the form *s* and *t*. K must be of algebraic type, as recognised by Maple. *K* must be of convolution type.

over - Maple has a built in Laplace transform method for application to integrals of the form,

$$\int_0^t K(t-s)f(s)ds.$$

However, this is only true for simple convolution kernels. For more complex kernels the third parameter should be entered as *over*. See section 10 an explanation of this.

Output parameters.

Equation - The entered equation.

Sol - The solution to the equation.

5. Error messages.

ERROR, code expects its first/second argument to be of type algebraic.

The forcing function G and the kernel K must be of algebraic type. If not then this error message is encountered.

ERROR, Laplace transforms cannot be applied to the specified function g.

The function g cannot be converted using Laplace transforms and so the solution method also cannot be applied. If such an error message occurs a solution should be attempted using the *voltres* routine, where the solution is expressed as a resolvent kernel equation.

ERROR, Laplace transforms cannot be applied to the specified kernel.

The function K cannot be transformed using Laplace transforms.

ERROR, The kernel is not of convolution type.

Maple does not recognise the integral to be of convolution type. If the user is certain the kernel is of convolution type, the third parameter can be entered to override the error.

6. Accuracy.

There are no accuracy measurements.

7. Further comments.

No further comments

8. Examples.

Example 1. Solve the linear convolution equation,

$$f(t) = t^2 + \frac{1}{2}\int_0^t (t-s)f(s)ds.$$

Listing of inputs and outputs for execution of problem.

```
> with(volterra):

> voltlap(t,(1/2)*(t-s));
```

Equation entered: , $f(t) = t^2 + \frac{1}{2}\int_0^t (t-s)f(s)ds.$

Solution of equation: , $f(t) = -4 + 4\cosh\left(\frac{1}{2}\sqrt{2}\,t\right).$

Example 2.

Solve,

$$f(t) = t + \frac{5}{2}\int_0^t \cosh(t-s)f(s)ds.$$

Listing of inputs and outputs for execution of problem.

```
> with(volterra):

> voltlap(t,5/2*cosh(t-s));
```

Equation entered: , $f(t) = t + \frac{5}{2}\int_0^t \cosh(t-s)f(s)ds.$

Solution of equation: , $f(t) = t - \frac{5}{2} - \frac{25}{82}e^{\frac{5}{4}t}\sinh\left(\frac{1}{4}\sqrt{41}t\right)\sqrt{41} + \frac{5}{2}e^{\frac{5}{4}t}\cosh\left(\frac{1}{4}\sqrt{41}t\right).$

9 Program listing.

#Laplace transform solution with simple forcing function.

```
restart;
readlib(laplace):
voltlap:=proc(G:algebraic,K:algebraic)
local LapK,Lapf,Sol,Equation;
LapK:=eval(subs(s=0,K));
#Check inputs have Laplace transforms.
if has(laplace(G,t,w),laplace) then ERROR(`Laplace transforms cannot be applied to
the specified function g.`);
elif has(laplace(K,t,w),laplace) then ERROR(`Laplace transforms cannot be applied to
the specified kernel.`);
elif not has(laplace(Int(K*f(s),s=0..t),t,w),laplace(f(t),t,w))=true and not nargs=3
then ERROR(`The kernel is not of convolution type.`);
fi;
#Compute solution.
Lapf:=laplace(G,t,w)/(1-laplace(LapK,t,w));
Lapf:=simplify(Lapf);
Sol:=invlaplace(Lapf,w,t);
Sol:=simplify(Sol);
Equation:=G+Int(K*f(s),s=0..t);
#Print results.
print(`Equation entered:-    `. f(t)=Equation);
print(`Solution of equation:-    `. f(t)=Sol);
end;
```

10 Programming problems and effectiveness of Maple.

Maple's own Laplace transform routine proved inadequate when attempting to transform the entire integral and so the chosen method of solution treats the kernel separately from the integral.

'voltone' - Maple routine.

1. Purpose.

voltone produces a solution to a linear Volterra integral equation of the first kind with a kernel of convolution type by the application of Laplace transforms.

2. Specification.

with(volterra):
voltone(G,K,over);

Algebraic expressions:- G, K.
optional string:- over.

3. Description.

voltone will produce a solution to a linear Volterra equation of the first kind with a convolution kernel of the form,

$$g(t) = \alpha \int_0^t K(t-s)f(s)ds, \quad 0 \le t \le T.$$

The kernel of the equation must be of convolution type so that the following Laplace transform theorem can be applied.

$$\ell\{K(t) * f(t)\} = \int_0^t K(t-s)f(s)ds.$$

The forcing function g must also have a Laplace transform, otherwise the equation is best suited to the *voltres* routine, which expresses the solution in resolvent kernel form. The solution method of this routine can be seen in chapter four, §4.3.2.

4. Parameters.

Input parameters.

G - The forcing function of the equation. If this function contains an independent variable it must be of the form *t*. G must be of algebraic type, as defined and recognised by Maple. The function must have a Laplace transform (see above).

K - The kernel of the equation. The independent variables of the equation must be of the form *s* and *t*. K must be of algebraic type, as recognised by Maple. The kernel must be of convolution type.

over - Maple has a built in Laplace transform method for application to integrals of the form,

$$\int_0^t K(t-s)f(s)ds.$$

However, this only works for simple convolution kernels. For more complex kernels the third parameter should be entered as *over*. See section 10 for more information.

Output parameters.

Equation - The entered equation.

Sol - The solution to the equation.

5. Error messages.

ERROR, voltone expects its first/second argument to be of type algebraic.

The forcing function G and the kernel K must be of algebraic type. If not then this error message is encountered.

ERROR, since g(0) does not equal zero, a solution does not exist.

The routine checks the forcing function is equal to zero. If the forcing function is not equal to zero a solution is not possible.

ERROR, Laplace transforms cannot be applied to the specified function g.

The function g cannot be converted using Laplace transforms and so the solution method cannot be applied. If such an error message occurs a solution should be attempted using the *voltres* routine, where the solution is expressed as a resolvent kernel equation.

ERROR, Laplace transforms cannot be applied to the specified kernel.

The function K cannot be transformed using Laplace transforms.

ERROR, The kernel is not of convolution type.

Maple does not recognise the integral to be of convolution type. If the user is certain the kernel is of convolution type, the third parameter can be entered to override the error.

Maple could not verify the solution of the equation.

The routine's solution checker contains code that allows Maple to evaluate the integral with the solution. Under some conditions Maple will not evaluate the integral and the solution can not be verified. (See section 6 and section 10).

The solution has been checked and found to be erroneous.

First kind equations are generally of an ill-posed nature. The routine contains a solution checker and if the computed solution is not valid in the given equation this error message will be encountered.

6. Accuracy.

Linear Volterra integral equations of the first kind are most often of an ill-posed nature and so the routine contains a solution verification algorithm where the solution is substituted into the original equation and checked against the forcing function. A numerical checking approach was chosen in preference to an analytical comparison technique because Maple can encounter problems when comparing complex analytical expressions. (See section 10). If the solution contains the Dirac delta function it will not be verified by the solution checker.

7. Further comments.

No further comments.

8. Examples.

Example 1. Solve the linear convolution equation of the first kind,

$$\frac{1}{2}e^{2t} - 2\cos(t) + t^2 + \frac{3}{2} = \frac{1}{\alpha}\int_0^t \cos(t-s)f(s)ds.$$

Listing of inputs and outputs for execution of problem.

```
> with(volterra):

> voltone((1/2)*exp(2*t)-2*cos(t)+t^2+(3/2),(1/alpha)*cos(t-s));
```

Equation entered: , $\frac{1}{2}e^{2t} - 2\cos(t) + t^2 + \frac{3}{2} = \int_0^t \frac{\cos(t-s)f(s)}{\alpha} ds.$

Solution of equation: , $f(t) = -\frac{1}{4}\alpha + \frac{7}{2}t\alpha + \frac{1}{3}t^3\alpha + \frac{5}{4}\alpha e^{2t}$.

Example 2.

Solve,

$$\frac{16}{15}\cos(t)^2 - 2e^{3t} + \frac{14}{15}e^{\frac{1}{2}t} = \int_0^t (\cos(t-s) + 2\cosh(t-s))f(s)ds.$$

Listing of inputs and outputs for execution of problem.

```
> with(volterra):

> voltone(16/15*cos(t)^2-2*exp(3*t)+14/15*exp(1/2*t),cos(t-s)+2*cosh(t-s));
```

Equation entered: , $\frac{16}{15}\cos(t)^2 - 2e^{3t} + \frac{14}{15}e^{\frac{1}{2}t} = \int_0^t (\cos(t-s) + 2\cosh(t-s))f(s)ds.$

Solution of equation: ,

$$f(t) = -\frac{18}{15}t + \frac{6}{5} - \frac{40}{21}e^{3t} - e^{\frac{1}{2}t} - \frac{4}{11}\sin(2t) + \frac{324}{385}\sin\left(\frac{1}{3}\sqrt{3}t\right)\sqrt{3} - \frac{44}{315}\cos\left(\frac{1}{3}\sqrt{3}t\right).$$

Maple could not verify the solution of the equation.

9 Program listing.

```
#Laplace transform solution of 1st kind Volterra.

restart;
readlib(laplace):
voltone:=proc(G:algebraic,K:algebraic)
local Solcheck,LapK,LapG,LapK2,InvFunc,Sol,Equation,Solsub,Right,sing,
Num_Right,Num_G,Difference;
#Check existence of solution.
Solcheck:=eval(subs(t=0,G));
if not Solcheck=0 then
ERROR(`Since g(0) is not equal to zero, a solution does not exist.`);
fi;
LapK:=eval(subs(s=0,K));
#Check for Laplace transforms.
if not (has(K,t) and has(K,s)) then
ERROR(`Variables in the given functions must be of type s and t.`)
elif has(laplace(G,t,w),laplace) then
ERROR(`Laplace transforms cannot be applied to the specified function g.`)
elif has(laplace(K,t,w),laplace) then
ERROR(`Laplace transforms cannot be applied to the specified kernel.`)
elif not has(laplace(Int(K*f(s),s=0..t),t,w),laplace(f(t),t,w)) and not nargs=3 then
ERROR(`The kernel is not of convolution type.`)
fi;
#Compute solution.
LapK:=eval(subs(s=0,K));
```

```
LapG:=simplify(laplace(G,t,w));
LapK2:=simplify(laplace(LapK,t,w));
InvFunc:=simplify(LapG/LapK2);
Sol:=invlaplace(InvFunc,w,t);
Sol:=simplify(Sol);
Equation:=Int(K*f(s),s=0..t);
print(`Equation entered:-  `, G=Equation);
print(`Solution of equation:-  `. f(t)=Sol);
#Verify solution.
Solsub:=eval(subs(t=s,Sol));
sing:=traperror(value(Int(K*Solsub,s=0..t)));
if sing=lasterror then
print(`Maple could not verify the solution of the equation.`)
else
Right:=value(Int(K*Solsub,s=0..t));
if not has(Solsub,Dirac) then
for i from -2 to 2 do
Num_Right:=evalf(subs(t=i,Right),15);
Num_G:=evalf(subs(t=i,G),15);
Difference:=Num_Right-Num_G;
if (Difference^2)>(1*10^(-10)) then
print(`This solution has been checked and found to be erroneous.`);
break;
fi;
od;
fi;
fi;
end;
```

10 Programming problems and effectiveness of Maple.

Maple's own Laplace transform routine proved inadequate when attempting to program this solution method. The command was unreliable when applied to convolution integrals, possibly due to memory restrictions, and so it is used only as a check for a convolution kernel. As a Laplace transform is only required of the forcing function and the kernel, applying Laplace transforms to the entire integral is not essential.

The routine contains a solution checker which works by numerical comparison of the forcing function and the integral. The integral is the given kernel and the computed solution which is evaluated by Maple's *value* command. When the integral becomes complex the *value* command will not return an explicit result as can be seen in example 2 above. The solution given in the example is correct but Maple could not evaluate the following integral,

$$\int_0^t (\cos(t-s)+2\cosh(t-s))*(-\frac{18}{15}s+\frac{6}{5}-\frac{40}{21}e^{3s}-e^{\frac{1}{2}s}-\frac{4}{11}\sin(2s)+\frac{324}{385}\sin\left(\frac{1}{3}\sqrt{3}t\right)\sqrt{3}-\frac{44}{315}\cos\left(\frac{1}{3}\sqrt{3}t\right))ds,$$

and so the solution checker could not verify the solution.

The final limitation when using Maple for this solution method is again contained in the solution checker. Initially, an analytical verification method was coded, but Maple could not recognise functions such as,

$$\frac{16}{15}\cos(t)^2 - \frac{8}{15},$$

to be the same as,

$$\frac{16}{15}\cos(2t),$$

without a lot of manipulation, and thus a numerical solution verification method was chosen.

'voltres' - Maple routine.

1. Purpose.

voltres will express the solution to a linear Volterra integral equation of the second kind in its resolvent kernel form. Laplace transforms will be applied to the convolution kernel.

2. Specification.

with(volterra):
voltres(G,K,over);

Algebraic expressions:- G, K.
optional string:- over.

3. Description.

voltres will apply Laplace transforms to the convolution kernel of a linear Volterra equation of the second kind, of the form,

$$f(t) = g(t) + \int_0^t K(t-s)f(s)ds, \quad 0 \le t \le T.$$

The kernel of the equation must be of convolution type so that the following Laplace transform theorem can be applied.

$$\ell\{K(t)*f(t)\} = \int_0^t K(t-s)f(s)ds.$$

The forcing function *g* must be of complex form where Laplace transforms cannot be applied, otherwise the equation is best suited to the *voltlap* routine. The solution method of this routine can be seen in chapter four, §4.3.1.

4. Parameters.

Input parameters.

G - The forcing function of the equation. If this function contains an independent variable it must be of the form *t*. G must be of algebraic type, as defined and recognised by Maple. The function should be of a complex form and one that does not have a Laplace transform (see above).

K - The kernel of the equation. The independent variables of the equation must be of the form *s* and *t*. K must be of algebraic type, as recognised by Maple. It must be of convolution type.

over - Maple has a built in Laplace transform method for the application to integrals of the form,

$$\int_0^t K(t-s)f(s)ds.$$

However, this is only true for simple convolution kernels. For more complex kernels the third parameter should be entered as *over*. See section 10 for more information.

Output parameters.

Equation - The entered equation.

Sol - The solution to the equation.

5. Error messages.

ERROR, voltres expects its first/second argument to be of type algebraic.

The forcing function G and the kernel K must be of algebraic type. If they are not then this error message is encountered.

ERROR, Laplace transforms can be applied to the specified function g. Use voltlap routine.

The function g does have a Laplace transform and the *voltres* routine is for complex forcing functions. The *voltlap* routine should be used if the forcing function has a Laplace transform.

ERROR, Laplace transforms cannot be applied to the specified kernel.

The function K cannot be transformed using Laplace transforms.

ERROR, The kernel is not of convolution type.

Maple does not recognise the integral to be of convolution type. If the user is certain the kernel is of convolution type the third parameter can be entered to override the error.

6. Accuracy.

There are no accuracy measurements.

7. Further comments.

No further comments

8. Examples.

Example 1. Solve the linear convolution equation,

$$f(t) = \frac{1}{\sec(t)^2} + \int_0^t \sinh(t-s) f(s) ds.$$

Listing of inputs and outputs for execution of problem.

```
> with(volterra):

> voltres(1/sec(t)^2,sinh(t-s));
```

Equation entered: , $f(t) = \frac{1}{\sec(t)^2} + \int_0^t \sinh(t-s) f(s) ds.$

Resolvent Kernel: , $\frac{1}{2}\sinh\left(\sqrt{2}t\right)\sqrt{2}$

Solution of equation: , $f(t) = \frac{1}{\sec(t)} + \int_0^t \frac{1}{2}\sinh\left(\sqrt{2}(t-s)\right)\sqrt{2} g(s) ds.$

Example 2.

Solve,

$$f(t) = \tan(t) + \sqrt{t+1} - \frac{1}{\alpha}\int_0^t (t-s) f(s) ds.$$

Listing of inputs and outputs for execution of problem.

```
> with(volterra):

> voltres(tan(t)+sqrt(t+1),1/alpha*(t-s));
```

Equation entered: , $f(t) = \tan(t) + \sqrt{t+1} - \frac{1}{\alpha}\int_0^t (t-s) f(s) ds.$

Resolvent Kernel: , $-\frac{\sin\left(\frac{t}{\sqrt{\alpha}}\right)}{\sqrt{\alpha}}$

Solution of equation: , $f(t) = \tan(t) + \sqrt{t+1} + \int_0^t -\frac{\sin\left(\frac{t-s}{\sqrt{\alpha}}\right) g(s)}{\sqrt{\alpha}} ds.$

9 Program listing.

 # Laplace solution of 2nd Volterra with complex g.

```
restart;
readlib(laplace):
voltres:=proc(G:algebraic,K:algebraic)
local LapK, Lapf, Res, Sol;
#Check parameters.
if not (has(K,s) and has(K,t)) then
ERROR(`Variables in the given functions must be of type s and t.`)
elif not has(laplace(G,t,w),laplace) then
ERROR(`Laplace transforms can be applied to the function g. Use voltlap routine.`)
elif has(laplace(K,t,w),laplace) then
ERROR(`Laplace transforms cannot be applied to the kernel.`)
elif not (has(laplace(Int(K*f(s),s=0..t),t,w),laplace(f(t),t,w)) or nargs=3) then
ERROR(`Kernel is not of convolution type.`)
fi;
#Compute resolvent kernel.
LapK:=subs(s=0,K);
Lapf:=laplace(LapK,t,w)/(1-laplace(LapK,t,w));
Lapf:=simplify(Lapf);
Res:=invlaplace(Lapf,w,t);
Res:=simplify(Res);
#Print solution.
print(`Equation entered:-    `. f(t)=G+Int(K*f(s),s=0..t));
print(`Resolvent `. Kernel=Res);
Res:=subs(t=w,Res);
Sol:=subs(w=(t-s),Res);
print(`Solution of equation:-    `. f(t)=G+Int(Sol*g(s),s=0..t));
end;
```

'volttrap' - Maple routine.

1. Purpose.

volttrap is a trapezium rule numerical approximation for linear Volterra equations of the second kind.

2. Specification.

```
with(volterra):
volttrap(G,K,h,n);
```

Algebraic expressions:- G, K.
Real positive constants:- h.
Positive integers:- n.

3. Description.

volttrap will apply the trapezium numerical approximation to a linear Volterra equation of the second kind of the form,

$$f(t) = g(t) + \int_0^t K(t,s)f(s)ds, \quad 0 \le t \le T.$$

The stepsize and the number of strips required for the routine are given as parameters by the user. Each approximate value is stored in a two dimensional array which is converted to list form and can then be used in conjunction with Maple's *plot* command to produce a graph of the solution. The solution method for this routine can be seen in chapter five, §5.2.1.

4. Parameters.

Input parameters.

G - The forcing function of the equation. If this function contains an independent variable it must be of the form *t*. G must be of algebraic type, as defined and recognised by Maple. *G* and *K* should not contain singularities.

K - The kernel of the equation. The independent variables of the equation must be of the form *s* and *t*. K must be of algebraic type, as recognised by Maple. It must be of convolution type.

h - The stepsize for the approximation to take. This parameter should be of positive real type and be less than one.

n - The number of strips over which the approximation is performed. *n* must be of positive integer type.

Output parameters.

Equation - The entered equation.

Sol - The solution to the equation.

results - The solution is contained in a list form and can be plotted by typing *plot(results);*.

5. Error messages.

ERROR, volttrap expects its first/second argument to be of type algebraic.

The forcing function *G* and the kernel *K* must be of algebraic type. If not then this error message is encountered.

ERROR, volttrap expects its third argument, h, to be of type positive.

The stepsize value h should be a real positive number otherwise this error shall occur.

ERROR, volttrap expects its fourth argument, n, to be of type posint.

The value of n (the number of strips for the approximation) must be a positive integer.

ERROR, stepsize is too large try less than 1.

Some functions will remain unevaluated if a large stepsize is entered. A stepsize of less than one is recommended.

ERROR, (in volttrap) division by zero.

When the problem contains singularities this error message will occur.

6. Accuracy.

There are no accuracy measurements.

7. Further comments.

The approximations are stored in a two dimensional array, one for the numerical approximation and one for the value of t. Once the approximation is complete the array is converted into list form so it can be utilised by the plot command. When using this command, a plot of f(t) will be given against t. Floating point arithmetic is used in the computations.

8. Examples.

Example 1. Apply the trapezium numerical solution method to the following equation,

$$f(t) = \frac{\sqrt{t+1}}{\ln(t+2)} - \int_0^t \left(t^{\frac{2}{3}} - \cosh\left(\frac{1}{2}t\right)\right) f(s)ds.$$

This equation cannot be solved using any of the previous analytical solution methods and is therefore chosen for the following three numerical solution techniques. The forcing function of this equation has been chosen to avoid singularities. Below are listings of inputs and outputs for execution of the routine.

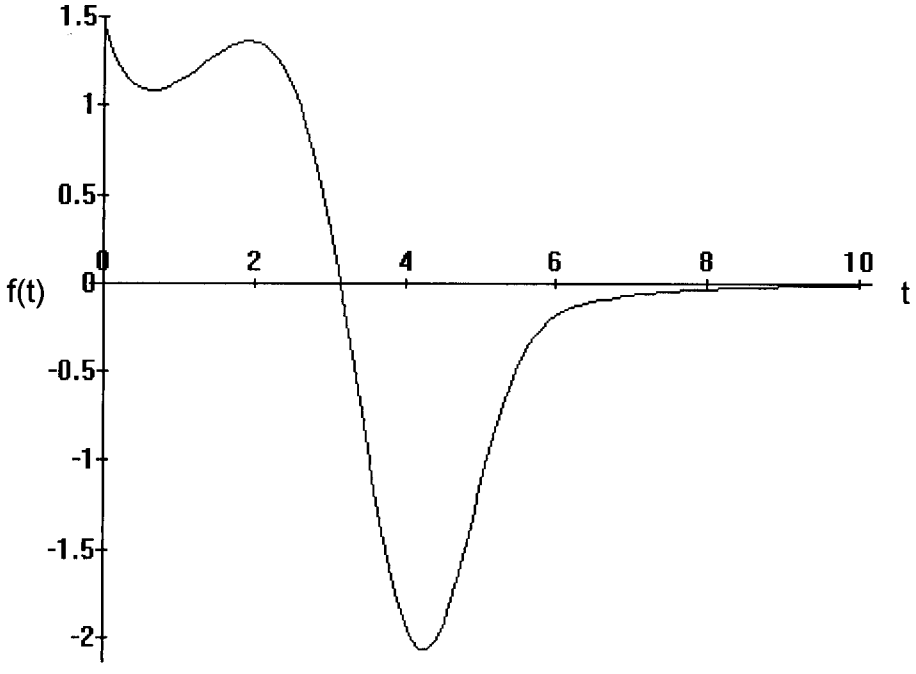
```
> with(volterra):

> volttrap(sqrt(t+1)/ln(t+2),t^(2/3)-cosh(t/2),0.1,100);
```

Equation entered: , $f(t) = \frac{\sqrt{t+1}}{\ln(t+2)} - \int_0^t \left(t^{\frac{2}{3}} - \cosh\left(\frac{1}{2}t\right)\right) f(s)ds.$

Approximate solution: f(t)= -0.0100713304.

```
> plot(results);
```

(The solution for this problem can be seen over the page).



Solution plot for f(t).

9 Program listing.

 # Trapezium approximation of Volterra second order linear.

```
restart;
volttrap:=proc(G:algebraic,K:algebraic,h:positive,n:posint)
local Func,P1Func1,P2Func1,i,P1Funci,P2Funci,j,m;
#Check inputs.
if h=1 or h>1 then
ERROR(`stepsize is too large, try less than 1.`)
fi;
#Trapezium rule approximation.
Func:=array(0..n,0..1);
Func[0,1]:=evalf(subs(t=0,G));
P1Func1:=eval(subs(t=h,G))+(h/2)*(eval(subs(t=h,s=0,K)))*Func[0,1];
P2Func1:=1-(h/2)*(eval(subs(t=h,s=h,K)));
Func[1,1]:=evalf(P1Func1/P2Func1);
for i from 2 to n do
P1Funci:=eval(subs(t=i*h,G))+(h/2)*(eval(subs(t=i*h,s=0,K)))*Func[0,1];
P2Funci:=0;
for j from 1 to (i-1) do
P2Funci:=P2Funci+eval(subs(t=i*h,s=j*h,K))*Func[j,1];
od;
P3Funci:=1-(h/2)*eval(subs(t=i*h,s=i*h,K));
Func[i,1]:=evalf((P1Funci+h*P2Funci)/P3Funci);
```

```
od;
#Create list for plots.
for m from 0 to n do
Func[m,0]:=m*h;
od;
results:=convert(Func,list,list);
Equation:=G+Int(K*f(s),s=0..t);
Sol:=Func[n,1];
#Print results.
print(`Equation entered:-    `. f(t)=G+Int(K*f(s),s=0..t));
print(`Approximate solution:-  `. f(t)=Func[n,1]);
end;
```

'voltsimp' - Maple routine.

1. Purpose.

voltsimp is a numerical approximation for linear Volterra equations of the second kind using Simpson's rule and applying Simpson's 3/8ths rule to the latter points of the interval.

2. Specification.

```
with(volterra):
voltsimp(G,K,h,n);
```

Algebraic expressions:- G, K.
Real positive constants:- h.
Positive integers:- n.

3. Description.

voltsimp will apply Simpson's numerical approximation to a linear Volterra equation of the second kind of the form,

$$f(t) = g(t) + \int_0^t K(t,s)f(s)ds, \quad 0 \le t \le T.$$

Simpson's rule is applied to all but the last four points of the interval, where Simpson's 3/8ths is applied. The stepsize and the number of strips required for the routine are given as parameters by the user. Each approximate value is stored in a two dimensional array which is converted to list form and can be used in conjunction with Maple's *plot* command to produce a graph of the solution. The solution method of this routine can be seen in chapter five, §5.2.2.

4. Parameters.

Input parameters.

G - The forcing function of the equation. If this function contains an independent variable it must be of the form *t*. G must be of algebraic type, as defined and recognised by Maple. The function should not contain singularities.

K - The kernel of the equation. The independent variables of the equation must be of the form *s* and *t*. K must be of algebraic type, as recognised by Maple. It must be of convolution type. The function should not contain singularities.

h - The stepsize for the approximation to take. This parameter should be of positive real type and be less than one.

n - The number of strips over which the approximation is performed. *n* must be of positive integer type.

Output parameters.

Equation - The entered equation.

Sol - The solution to the equation.

results - The solution is contained in a list form and can be plotted by typing *plot(results);*.

5. Error messages.

ERROR, voltsimp expects its first/second argument to be of type algebraic.

The forcing function G and the kernel K must be of algebraic type. If not then this error message is encountered.

ERROR, voltsimp expects its third argument, h, to be of type positive.

The stepsize value h should be a real positive number, otherwise this error will occur.

ERROR, voltsimp expects its fourth argument, n, to be of type posint.

The value of n (the number of strips for the approximation) must be a positive integer.

ERROR, stepsize is too large, try less than 1.

Some functions will remain unevaluated if a large stepsize is entered. A stepsize of less than one is recommended.

ERROR, number of strips must be two or more.

Simpson's rule requires at least two strips (the parameter n), to work.

ERROR, (in voltsimp) division by zero.

When the problem contains singularities this error message will occur.

6. Accuracy.

There are no accuracy measurements.

7. Further comments.

Simpson's rule requires one starting value. This starting value is computed using the Trapezium rule. As the observed error for Simpson's 3/8ths is approximately $O(h^5)$ and $O(h^2)$ for the trapezium rule, a stepsize of $\sqrt{h^5}$ is used for the starting value approximation to maintain the observed error of $O(h^5)$. The approximations are stored in a two dimensional array, one for the numerical approximation and one for the value of t. Once the approximation is complete, the array is converted into list form so it can be utilised by the plot command. When using the *plot* command, a plot of f(t) will be given against t. Floating point arithmetic is used in the computations.

8. Examples.

Example 1. Apply Simpson's 3/8ths numerical solution method to the following equation,

$$f(t)=\frac{\sqrt{t+1}}{\ln(t+2)}-\int_0^t\left(t^{\frac{2}{3}}-\cosh\left(\frac{1}{2}t\right)\right)f(s)ds.$$

The forcing function of this equation has been chosen to avoid singularities. Listing of inputs and outputs for execution of problem.

```
> with(volterra):

> voltsimp(sqrt(t+1)/ln(t+2),t^(2/3)-cosh(t/2),0.1,100);
```

Equation entered: , $f(t)=\frac{\sqrt{t+1}}{\ln(t+2)}-\int_0^t\left(t^{\frac{2}{3}}-\cosh\left(\frac{1}{2}t\right)\right)f(s)ds.$

Approximate solution: f(t)= -0.02024337055.

9 Program listing.

#Simpson's 3/8 solution method.

```
restart;
voltsimp:=proc(G:algebraic,K:algebraic,h:positive,n:posint)
local Func0,P1Func1,P2Func1,Func1,Func,i,P1Funci,P2Funci,h2,j,weight,
P3Funci,w,endweight;
#Check inputs.
if h=1 or h>1 then
ERROR(`stepsize is too large, try less than 1.`)
fi;
if n<2 then
ERROR(`Number of strips must be two or more.`)
fi;
Func0:=evalf(subs(t=0,G));
#Find f1 approx by trapezium rule.
h2:=evalf(sqrt(h^5));
P1Func1:=eval(subs(t=h2,G))+(h2/2)*(eval(subs(t=h2,s=0,K)))*Func0;
P2Func1:=1-(h2/2)*(eval(subs(t=h2,s=h2,K)));
Func1:=evalf(P1Func1/P2Func1);
#Simpsons approximation.
Func:=array(0..n,0..1);
Func[0,1]:=Func0;
Func[1,1]:=Func1;
for i from 2 to n do
if i=3 then
Func[3,1]:=evalf((eval(subs(t=3*h,G))+h*((3/8)*eval(subs(t=3*h,s=0,K))*Func[0,1]+
(9/8)*eval(subs(t=3*h,s=h,K))*Func[1,1]+(9/8)*eval(subs(t=3*h,s=2*h,K))*Func[2,1
]))/(1-h*(3/8)*eval(subs(t=3*h,s=3*h,K)))) else
P1Funci:=eval(subs(t=i*h,G))+h*((1/3)*eval(subs(t=i*h,s=0,K)))*Func0;
P2Funci:=0;
if type(i,even)=true then
for j from 1 to (i-1) do
if type(j,even)=true then weight:=2/3 else weight:=4/3 fi;
P2Funci:=P2Funci+h*weight*(eval(subs(t=i*h,s=j*h,K)))*Func[j,1];
od;
fi;
if type(i,odd)=true then
for j from 1 to (i-1) do
if type(j,even)=true then
if j<i-3 then weight :=2/3
elif j=i-3 then weight:=17/24
elif j=i-1 then weight:=9/8
fi;
elif j<i-3 then weight:=4/3
elif j=i-2 then weight:=9/8
fi;
```

```
P2Funci:=P2Funci+h*weight*eval(subs(t=i*h,s=j*h,K))*Func[j,1];
od;
fi;
if type(i,even)=true then endweight:=1/3 else endweight:=3/8 fi;
P3Funci:=h*endweight*eval(subs(t=i*h,s=i*h,K));
Func[i,1]:=evalf((P1Funci+P2Funci)/(1-P3Funci));
fi;
od;
#Create list for plots.
for w from 0 to n do
Func[w,0]:=h*w;
od;
results:=convert(Func,list,list);
#Print results.
Equation:=G+Int(K*f(s),s=0..t);
Sol:=Func[n,1];
print(`Equation entered:-    `.f(t)=Equation);
print(`Approximate solution:-    `.f(t)=Sol);
end;
```

'voltgreg' - Maple routine.

1. Purpose.

voltgreg is a numerical approximation for linear Volterra equations of the second kind using Gregory's fourth order approximation.

2. Specification.

with(volterra):
voltgreg(G,K,h,n);

Algebraic expressions:- G, K.
Real positive constants:- h.
Positive integers:- n.

3. Description.

voltgreg will apply Gregory's fourth order numerical approximation to a linear Volterra equation of the second kind of the form,

$$f(t) = g(t) + \int_0^t K(t,s)f(s)ds, \quad 0 \le t \le T.$$

Gregory's rule requires four starting values. The first is generated by the trapezium rule with a stepsize of h^2 and the remaining three by Simpson's rule. The stepsize and the number of strips required for the routine are given as parameters by the user. Each approximate value is stored in a two dimensional array which is converted to list form and can then be used in conjunction with Maple's *plot* command to produce a graph of the solution. The solution method of this routine can be seen in chapter five, §5.2.3.

4. Parameters.

Input parameters.

G - The forcing function of the equation. If this function contains an independent variable it must be of the form *t*. G must be of algebraic type, as defined and recognised by Maple. The function should not contain singularities.

K - The kernel of the equation. The independent variables of the equation must be of the form *s* and *t*. K must be of algebraic type, as recognised by Maple. *K* must be of convolution type. The function should not contain singularities.

h - The stepsize for the approximation to take. This parameter should be of positive real type and be less than one.

n - The number of strips over which the approximation is performed. *n* must be of positive integer type, and greater than four.

Output parameters.

Equation - The entered equation.

Sol - The solution to the equation.

results - The solution is contained in a list form and can be plotted by typing *plot(results);*.

5. Error messages.

ERROR, voltgreg expects its first/second argument to be of type algebraic.

The forcing function G and the kernel K must be of algebraic type. If not then this error message is encountered.

ERROR, voltgreg expects its third argument, h, to be of type positive.

The stepsize value h should be a real positive number, otherwise this error will occur.

ERROR, voltgreg expects its fourth argument, n, to be of type posint.

The value of n (the number of strips for the approximation) must be a positive integer.

ERROR, stepsize is too large, try less than 1.

Some functions will remain unevaluated if a large stepsize is entered. A stepsize of less than one is recommended.

ERROR, number of strips must be five or more.

Gregory's rule requires at least five strips (the parameter n) to work.

ERROR, (in voltgreg) division by zero.

When the problem contains singularities this error message will occur.

6. Accuracy.

There are no accuracy measurements.

7. Further comments.

The approximations are stored in a two dimensional array, one for the numerical approximation and one for the value of t. Once the approximation is complete the array is converted into list form so that a graph can be produced using the plot command. This gives a plot of f(t) against t. Floating point arithmetic is used in the computations.

8. Examples.

Example 1. Apply Gregory's fourth order numerical approximation method to the following equation,

$$f(t)=\frac{\sqrt{t+1}}{\ln(t+2)}-\int_0^t\left(t^{\frac{2}{3}}-\cosh\left(\frac{1}{2}t\right)\right)f(s)ds.$$

Listing of inputs and outputs for execution of problem.

```
> with(volterra):

> voltgreg(sqrt(t+1)/ln(t+2),t^(2/3)-cosh(t/2),0.1,100);
```

Equation entered: , $f(t)=\frac{\sqrt{t+1}}{\ln(t+2)}-\int_0^t\left(t^{\frac{2}{3}}-\cosh\left(\frac{1}{2}t\right)\right)f(s)ds.$

Approximate solution: f(t)= -0.01006742133.

9 Program listing.

```
#Gregory's solution method.

restart;
```

```
voltgreg:=proc(G:algebraic,K:algebraic,h:positive,n:posint)
local Func0,
h2,P1Func1,P2Func1,Func1,P1Func2,P2Func2,Func2,P1Func3,P2Func3,
Func3,P1Func4,P2Func4,Func4,Func,i,P1Funci,P2Funci,P3Funci,w;
#Check inputs.
if (h=1 or h>1) then
ERROR(`stepsize is too large, try less than 1.`);
fi;
if n<5 then
ERROR(`Number of strips must be five or more.`);
fi;
Func0:=evalf(subs(t=0,G));
#Find f1 approximation using trapezium rule.
h2:=h^2;
P1Func1:=eval(subs(t=h2,G))+(h2/2)*(eval(subs(t=h2,s=0,K)))*Func0;
P2Func1:=1-(h2/2)*(eval(subs(t=h2,s=h2,K)));
Func1:=evalf(P1Func1/P2Func1);
#Find f2, f3, f4 using Simpson's rule.
#Find f2 appromination.
P1Func2:=((h/3)*(eval(subs(t=2*h,s=0,K)))*Func0)+((h*4/3)*(eval(subs(t=2*h,s=h,
K)))*Func1);
P2Func2:=1-(h/3)*eval(subs(t=2*h,s=2*h,K));
Func2:=(eval(subs(t=2*h,G))+P1Func2)/P2Func2;
Func2:=evalf(Func2);
#Find f3 approximation.
P1Func3:=eval(subs(t=3*h,G))+h*(((3/8)*eval(subs(t=3*h,s=0,K))*Func0)+((9/8)*ev
al(subs(t=3*h,s=h,K))*Func1)+((9/8)*eval(subs(t=3*h,s=2*h,K))*Func2));
P2Func3:=1-((3*h/8)*eval(subs(t=3*h,s=3*h,K)));
Func3:=P1Func3/P2Func3;
Func3:=evalf(Func3);
#Find f4 approximation.
P1Func4:=eval(subs(t=4*h,G))+h*(((1/3)*eval(subs(t=4*h,s=0,K))*Func0)+((4/3)*ev
al(subs(t=4*h,s=h,K))*Func1)+((2/3)*eval(subs(t=4*h,s=2*h,K))*Func2)+((4/3)*eval
(subs(t=4*h,s=3*h,K))*Func3));
P2Func4:=1-((h/3)*eval(subs(t=4*h,s=4*h,K)));
Func4:=evalf(P1Func4/P2Func4);
#The Gregory approximation.
Func:=array(0..n,0..1);
Func[0,1]:=Func0;
Func[1,1]:=Func1;
Func[2,1]:=Func2;
Func[3,1]:=Func3;
Func[4,1]:=Func4;
for i  from 5 to n do
P1Funci:=h*((3/8)*eval(subs(t=i*h,s=0,K))*Func[0,1]+(7/6)*eval(subs(t=i*h,s=h,K))
*Func[1,1]+(23/24)*eval(subs(t=i*h,s=2*h,K))*Func[2,1]);
P2Funci:=0;
for j from 3 to (i-3) do
```

```
P2Funci:=P2Funci+h*eval(subs(t=i*h,s=j*h,K))*Func[j,1];
od;
P3Funci:=h*((23/24)*eval(subs(t=i*h,s=(i*h-2*h),K))*Func[i-
2,1]+((7/6)*eval(subs(t=i*h,s=(i*h-h),K))*Func[i-1,1]));
Func[i,1]:=(eval(subs(t=i*h,G))+P1Funci+P2Funci+P3Funci)/(1-
(h*3/8*eval(subs(t=i*h,s=i*h,K))));
od;
Func[i,1]:=evalf(Func[i,1]);
#Convert to list form.
for w from 0 to n do
Func[w,0]:=h*w;
od;
results:=convert(Func,list,list);
Equation:=G+Int(K*f(s),s=0..t);
Sol:=Func[n,1];
#Print results.
print(`Equation entered:-    `. f(t)=Equation);
print(`Approximate solution:-    `. f(t)=Sol);
end;
```

'simpini' - Maple routine.

1. Purpose.

simpini is Simpson's 3/8ths numerical approximation method of a linear Volterra integral equation using an initial value.

2. Specification.

```
with(volterra):
simpini(G,K,fl,h,n);
```

Algebraic expressions:- G, K.
Real positive constants:- h.
Positive integers:- n.
Real constants:- fl.

3. Description.

simpini will apply Simpson's numerical approximation, with an initial starting value, to a linear Volterra equation of the second kind of the form,

$$f(t) = g(t) + \int_0^t K(t,s)f(s)ds, \quad 0 \le t \le T.$$

This routine is essentially identical to the *voltsimp* routine except the starting value is given by the user. The stepsize and the number of strips required for the routine are also given as parameters by the user. The initial starting value fl, should be a real number. Each approximate value is stored in a two dimensional array which is converted to list form and can then be used in conjunction with Maple's *plot* command to produce a graph of the solution. The solution method of this routine can be seen in chapter five, §5.2.2.

4. Parameters.

Input parameters.

G - The forcing function of the equation. If this function contains an independent variable it must be of the form *t*. G must be of algebraic type, as defined and recognised by Maple. The function should not contain singularities.

K - The kernel of the equation. The independent variables of the equation must be of the form *s* and *t*. K must be of algebraic type, as recognised by Maple. *K* must be of convolution type. The function should not contain singularities.

fl - The starting value. In the *voltsimp* routine the starting value is generated by the Trapezium rule. Here the user gives the starting value as a parameter and it must be of real type.

h - The stepsize for the approximation to take. This parameter should be of positive real type and be less than one.

n - The number of strips over which the approximation is performed. *n* must be of positive integer type and greater than four.

Output parameters.

Equation - The entered equation.

Sol - The solution to the equation.

results - The solution is contained in list form and can be plotted by typing *plot(results);*.

5. Error messages.

ERROR, simpini expects its first/second argument to be of type algebraic.

The forcing function G and the kernel K must be of algebraic type. If not then this error message is encountered.

ERROR, simpini expects its third argument, fl, to be of type realcons.

The initial value must be a real number.

ERROR, simpini expects its fourth argument, h, to be of type positive.

The stepsize value h should be a real positive number, otherwise this error will occur.

ERROR, simpini expects its fifth argument, n, to be of type posint.

The value of n (the number of strips for the approximation) must be a positive integer.

ERROR, stepsize is too large, try less than 1.

Some functions will remain unevaluated if a large stepsize is entered. A stepsize of less than one is recommended.

ERROR, number of strips must be two or more.

Simpson's rule requires at least two strips (the parameter n), to work.

ERROR, (in simpini) division by zero.

When the problem contains singularities this error message will occur.

6. Accuracy.

The example given below in section 8 has an exact solution of

$$f(t)=\frac{1}{2}t+\frac{1}{2}-\frac{1}{10}e^{-\frac{1}{2}t}\sinh\left(\frac{1}{2}\sqrt{5}t\right)\sqrt{5}-\frac{1}{2}e^{-\frac{1}{2}t}\cosh\left(\frac{1}{2}\sqrt{5}t\right),$$

and so by using a starting value of f(0.1)=0.0475792883, the routine produces an approximation of -169.3076124. Comparing this to the exact solution value of f(10) = -169.3075368, giving an observed error of -7.56×10^{-5}, it can be seen that the solution method has an accuracy of approximately $O(h^5)$.

7. Further comments.

The approximations are stored in a two dimensional array, one for the numerical approximation and one for the value of t. Once the approximation is complete the array is converted into list form so it can be used with the plot command. When using the *plot* command, a plot of f(t) will be given against t. Floating point arithmetic is used in the computations.

8. Examples.

Example 1. Apply Simpson's numerical approximation method to the following equation,

$$f(t)=\frac{1}{2}t-\int_0^t\cosh(t-s)f(s)ds.$$

using a starting value of 0.0475792883.

Listing of inputs and outputs for execution of problem.

> with(volterra):

> simpini(t/2,-cosh(t-s),0.0475792883,0.1,100);

Equation entered: , $f(t)=\frac{1}{2}t-\int_0^t \cosh(t-s)f(s)ds.$

Approximate solution: f(t)= -169.3076124.

9 Program listing.

#Simpson's 3/8 solution method, with starting values.

```
restart;
simpini:=proc(G:algebraic,K:algebraic,f1:realcons,h:positive,n:integer)
local P1Func1,P2Func1,Func1,Func,i,P1Funci,P2Funci,j,weight,
P3Funci,w,endweight;
#Check inputs.
if n<2 then
ERROR(`Number of strips must be two or more.`);
fi;
if h=1 or h>1 then
ERROR(`Stepsize is too large, try less than 1.`)
fi;
#Simpsons approximation.
Func:=array(0..n,0..1);
Func[0,1]:=evalf(subs(t=0,G));
Func[1,1]:=f1;
for i from 2 to n do
if i=3 then
Func[3,1]:=evalf(eval(subs(t=3*h,G))+h*((3/8)*eval(subs(t=3*h,s=0,K))*Func[0,1]+(
9/8)*eval(subs(t=3*h,s=h,K))*Func[1,1]+(9/8)*eval(subs(t=3*h,s=2*h,K))*Func[2,1]
))/(1-h*(3/8)*eval(subs(t=3*h,s=3*h,K))) else
P1Funci:=eval(subs(t=i*h,G))+h*((1/3)*eval(subs(t=i*h,s=0,K)))*Func[0,1];
P2Funci:=0;
if type(i,even)=true then
for j from 1 to (i-1) do
if type(j,even)=true then weight:=2/3 else weight:=4/3 fi;
P2Funci:=P2Funci+h*weight*(eval(subs(t=i*h,s=j*h,K)))*Func[j,1];
od;
fi;
if type(i,odd)=true then
for j from 1 to (i-1) do
if type(j,even)=true then
if j<i-3 then weight :=2/3
elif j=i-3 then weight:=17/24
elif j=i-1 then weight:=9/8
fi;
```

```
elif j<i-3 then weight:=4/3
elif j=i-2 then weight:=9/8
fi;
P2Funci:=P2Funci+h*weight*eval(subs(t=i*h,s=j*h,K))*Func[j,1];
od;
fi;
if type(i,even)=true then endweight:=1/3 else endweight:=3/8 fi;
P3Funci:=h*endweight*eval(subs(t=i*h,s=i*h,K));
Func[i,1]:=evalf((P1Funci+P2Funci)/(1-P3Funci));
fi;
od;
#Create list for plots.
for w from 0 to n do
Func[w,0]:=h*w;
od;
results:=convert(Func,list,list);
Equation:=G+Int(K*f(s),s=0..t);
Sol:=Func[n,1];
#Print results.
print(`Equation entered:-    `.f(t)=Equation);
print(`Approximate solution:-    `.f(t)=Sol);
end;
```

'gregory' - Maple routine.

1. Purpose.

gregory is Gregory's fourth order numerical approximation method of a linear Volterra integral equation using four initial values.

2. Specification.

with(volterra):
simpini(G,K,f1,f2,f3,f4,h,n);
Algebraic expressions:- G, K.
Real positive constants:- h.
Positive integers:- n.
Real constants:- f1, f2, f3, f4.

3. Description.

gregory will apply Gregory's fourth order numerical approximation with four initial starting values to a linear Volterra equation of the second kind of the form,

$$f(t) = g(t) + \int_0^t K(t,s)f(s)ds, \quad 0 \le t \le T.$$

This is fundamentally identical to the *voltgreg* routine except the starting values are given by the user. The stepsize and the number of strips required for the routine are also given as parameters by the user. The initial starting values, f1, f2, f3, and f4 should be real numbers. Each approximate value is stored in a two dimensional array which is converted to list form and can then be used in conjunction with Maple's *plot* command to produce a graph of the solution. The solution method of this routine can be seen in chapter five, §5.2.3.

4. Parameters.

Input parameters.

G - The forcing function of the equation. If this function contains an independent variable it must be of the form *t*. G must be of algebraic type, as defined and recognised by Maple. The function should not contain singularities.

K - The kernel of the equation. The independent variables of the equation must be of the form *s* and *t*. K must be of algebraic type, as recognised by Maple. It must be of convolution type. The function should not contain singularities.

f1, f2, f3, f4 - The starting values. In the *voltgreg* routine, the starting values are generated by the Trapezium rule and Simpson's rule. Here the user gives the starting values as parameters and they must be of real type.

h - The stepsize for the approximation to take. This parameter should be of positive real type and be less than one.

n - The number of strips over which the approximation is performed. *n* must be of positive integer type, and greater than four.

Output parameters.

Equation - The entered equation.

Sol - The solution to the equation.

results - The solution is contained in list form and can be plotted by typing *plot(results);*.

5. Error messages.

ERROR, gregory expects its first/second argument to be of type algebraic.

The forcing function G and the kernel K must be of algebraic type. If not then this error message is encountered.

ERROR, gregory expects its third/fourth/fifth/sixth argument, f1, f2, f3, f4, to be of type realcons.

The initial values must be a real number.

ERROR, gregory expects is seventh argument, h, to be of type positive.

The stepsize value h should be a real positive number, otherwise this error will occur.

ERROR, gregory expects its eighth argument, n, to be of type posint.

The value of n (the number of strips for the approximation) must be a positive integer.

ERROR, stepsize is too large, try less than 1.

Some functions will remain unevaluated if a large stepsize is entered. A stepsize of less than one is recommended.

ERROR, number of strips must be five or more.

Gregory's rule requires at least five strips (the parameter n) to work.

ERROR, (in gregory) division by zero.

When the problem contains singularities this error message will occur.

6. Accuracy.

Once more, by taking the example given below in section 8 whose exact solution is,

$$f(t)=\frac{1}{2}t+\frac{1}{2}-\frac{1}{10}e^{-\frac{1}{2}t}\sinh\left(\frac{1}{2}\sqrt{5}t\right)\sqrt{5}-\frac{1}{2}e^{-\frac{1}{2}t}\cosh\left(\frac{1}{2}\sqrt{5}t\right),$$

we can generate starting values of $f(0.1)=0.0475792883$, $f(0.2)=0.0906037877$, $f(0.3)=0.1294405074$ and $f(0.4)=0.1643816459$. The routine produces an approximation of -169.30696366. Comparing this to the exact solution value of $f(10)=-169.3075368$, giving an observed error of 5.7314×10^{-4}, it can be seen that the solution method has an accuracy of approximately $O(h^4)$.

7. Further comments.

The approximations are stored in a two dimensional array, one for the numerical approximation and one for the value of t. Once the approximation is complete the array is converted into list form so it can be used by the plot command. When using the *plot* command a plot of f(t) will be given against t. Floating point arithmetic is used in the computations.

8. Examples.

Example 1. Apply Gregory's fourth order numerical approximation method to the following equation,

$$f(t)=\frac{1}{2}t-\int_0^t\cosh(t-s)f(s)ds.$$

using a starting values of 0.0475792883, 0.0906037877, 0.1294405074 and 0.1643816459.

Listing of inputs and outputs for execution of problem.

```
> with(volterra):
> gregory(t/2,-cosh(t-s),0.04757929,0909060379,0.1294405,0.16438165,0.1,100);
```

$$\textit{Equation entered:}\ ,\ f(t)=\frac{1}{2}t-\int_0^t \cosh(t-s)f(s)ds.$$

$$\textit{Approximate solution:}\quad f(t)= -169.30696366.$$

9 Program listing.

```
#Gregory approximation using starting values.
restart;
gregory:=proc(G:algebraic,K:algebraic,f1:realcons,f2:realcons,f3:realcons,f4:realcons,
h:positive,n:posint)
local Func,i,P1Funci,P2Funci,P3Funci,w;
#Check inputs.
if n<5 then
ERROR(`Number of strips must be five or more.`);
fi;
Func:=array(0..n,0..1);
Func[0,1]:=evalf(subs(t=0,G));
Func[1,1]:=f1;
Func[2,1]:=f2;
Func[3,1]:=f3;
Func[4,1]:=f4;
#Gregory approximation.
for i  from 5 to n do
P1Funci:=h*((3/8)*eval(subs(t=i*h,s=0,K))*Func[0,1]+(7/6)*eval(subs(t=i*h,s=h,K))
*Func[1,1]+(23/24)*eval(subs(t=i*h,s=2*h,K))*Func[2,1]);
P2Funci:=0;
for j from 3 to (i-3) do
P2Funci:=P2Funci+h*eval(subs(t=i*h,s=j*h,K))*Func[j,1];
od;
P3Funci:=h*((23/24)*eval(subs(t=i*h,s=(i*h-2*h),K))*Func[i-
2,1]+((7/6)*eval(subs(t=i*h,s=(i*h-h),K))*Func[i-1,1]));
Func[i,1]:=evalf((eval(subs(t=i*h,G))+P1Funci+P2Funci+P3Funci)/(1-
(h*3/8*eval(subs(t=i*h,s=i*h,K)))));
od;
#Create list for plots.
for w from 0 to n do
Func[w,0]:=h*w;
od;
results:=convert(Func,list,list);
Equation:=G+Int(K*f(s),s=0..t);
Sol:=Func[n,1];
print(`Equation entered:-  `. f(t)=Equation);
print(`Approximate solution:-  `.f(t)=Sol);
end;
```

'voltneum' - Maple routine.

1. Purpose.

voltneum produces a solution to a linear Volterra integral equation of the second kind by applying the Neumann series iterative method.

2. Specification.

with(volterra):
voltneum(G,K,iter,C);

Algebraic expressions:- G, K.
Real positive constants:- iter (optional).
String:- C, (optional).

3. Description.

The routine *voltneum* applies the Neumann iterative approximation series to a linear Volterra integral equation of the second kind of the form,

$$f(t) = g(t) + \int_0^t K(t,s)f(s)ds, \quad 0 \le t \le T.$$

to produce a solution for f(t).

The integral of each iterative step is evaluated using Maple's *value* command and, providing an explicit expression can be obtained, a solution will be given for f(t). This solution for f(t) can then be compared with the solution for $f(t_{n-1})$ to determine whether the iterative series is convergent. The formula for this process is,

$$\left|f(t_n) - f(t_{n-1})\right| < \frac{1}{(b-a)},$$

where the functions $f(t_n)$ and $f(t_{n-1})$ are evaluated for $t = 0,1,3$. This method of evaluating convergence is preferred because comparisons of analytical results can be complex in Maple. See documentation for *codeone* routine, section 10.

The number of iterations performed by the routine can be assigned in the parameter inputs, but if no parameter is entered a default value of five will be applied. The solution method of this routine can be seen in chapter five, §5.1.

4. Parameters.

Input parameters.

G - The forcing function of the equation. If this function contains an independent variable it must be of the form *t*. G must be of algebraic type, as defined and recognised by Maple. The function should not contain singularities.

K - The kernel of the equation. The independent variables of the equation must be of the form *s* and *t*. K must be of algebraic type, as recognised by Maple. The function should not contain singularities.

iter - (optional). The number of iterations used in the Neumann series approximation. If no third parameter is entered a default value of five iterations shall be used.

C - (optional). If a check for convergence is desired the addition of this parameter will produce a numerical check.

Output parameters.

Equation - The entered equation.

Sol - The solution to the equation. In this routine the output parameter is defined as global so that it can be used with Maple's *plot* command after execution of the problem. It is often more useful to analyse a graph with this solution method as some results can be complex.

5. Error messages.

ERROR, voltneum expects its first/second argument to be of type algebraic.

The forcing function G and the kernel K must be of algebraic type. If not then this error message is encountered.

ERROR, variable for number of iterations must be of type (posint).

The optional parameter of *iter*, defining the desired number of iterations, must be a positive integer.

ERROR, Number of iterations must be three or more.

To obtain enough data to establish convergence, a minimum number of three iterations must be performed.

ERROR, (in voltneum) division by zero.

When the problem contains singularities this error message will occur.

ERROR, Unable to evaluate integral.

Maple's built in *value* command, used to solve the integral term of the iteration, cannot sufficiently reduce the integrand to exclude the integral sign.

ERROR, (in voltneum) cannot evaluate Boolean.

When checking for convergence, the routine compares the difference between the last two iterations against the values 0,1,3, (see section three). If Maple cannot sufficiently evaluate the condition this error will be returned.

6. Accuracy.

The routine performs a numerical check on the last two iterations to verify whether the computed solution is convergent.

7. Further comments.

The Neumann iterative approximation series routine works well with simple equations producing reasonable solutions after a number of iterations. However, when the kernel function becomes more complex, the evaluation of the integral can become restricted and the solution can be very lengthy. In such circumstances, use of the command *plot(Sol,t=0..n)* (where n is a positive integer) can present a clearer interpretation of the solutions behaviour.

8. Examples.

Example 1. Apply Neumann's iterative approximation series to the equation,

$$f(t) = 1 + \int_0^t f(s)ds.$$

This simple problem has been chosen to show how inept the Neumann series is at producing a solution for Volterra integral equations. With a routine such as *voltneum*, the task is made easier although interpretation of the solution remains arduous. For verification of convergence a minimum number of nine iterations is required.

A listing of inputs and outputs follows.

```
> with(volterra):

> voltneum(1,1,9,C);
```

Equation entered: , $f(t) = 1 + \int_0^t f(s)ds.$

Solution of equation:

$$f(t) = 1 + 0.0083333333t^5 + 0.166666667t^3 + 0.00002480158730t^8$$
$$+0.50000000t^2 + t + 0.00019841269841t^7 + 0.001388888889t^6 + 0.0416666667t^4$$

Neumann series approximation appears to converge.

Comparing this solution with the exact solution of e^t, at the point t=1, the error of the solution method is -0.3057×10^{-5}. Although this method might be said to produce 'clumsy' results, its accuracy is quite acceptable.

9 Program listing.

```
#Neumann series solution for Volterra.

restart;
voltneum:=proc(G:algebraic,K:algebraic)
local i,iter,FuncT;
iter:=5;
converge:=array(1..2);
Conv:=array(0..3);
#Check inputs.
if nargs=3 or nargs=4 then
if type(args[3],posint)=false then
ERROR(`Variable for number of iterations must be of type (posint).`);
fi;
if args[3]<3 then
ERROR(`Variable for the number of iterations must be three or more.`);
fi;
iter:=args[3];
fi;
FuncT:=G;
#Neumann iterative algorithm.
for i from 2 to iter do
FuncT:=eval(subs(t=s,FuncT));
sing:=traperror(value(G+Int(FuncT*K,s=0..t)));
if sing=lasterror then
ERROR(`Unable to evaluate integral.`);
else
FuncT:=value(G+Int(FuncT*K,s=0..t));
fi;
```

```
if i=(iter-1) then
converge[1]:=FuncT;
fi;
od;
converge[2]:=FuncT;
Sol:=simplify(evalf(FuncT));
if has(Sol,Int)=true then
ERROR(`Unable to evaluate integral.`);
fi;
#Print solution.
Equation:=G+Int(K*f(s),s=0..t);
print(`Equation entered:-  `, f(t)=Equation);
print(`Solution of equation:-  `, f(t)=Sol);
if nargs=4 then
#Check for convergence.
for i in [0,1,3] do
Conv[i]:=evalf(eval(subs(t=i,converge[2]-converge[1])));
Conv[i]:=sqrt(Conv[i]^2);
od;
if Conv[0]=0 and Conv[1]<1 and Conv[3]<1/3 then
print(`Neumann series approximation appears to converge.`);
else
print(`Neumann series approximation does not appear to converge.`);
fi;
fi;
end;
```

'firstmid' - Maple routine.

1. Purpose.

firstmid approximates a linear Volterra integral equation of the first kind using the mid-point rule. Richardson's extrapolation method can also be computed.

2. Specification.

with(volterra):
firstmid(G,K,h,n,R);

Algebraic expressions:- G, K.
Real positive constants:- h.
Positive integers:- n.
Optional string expression:- R.

3. Description.

The *firstmid* routine will apply the mid-point numerical approximation rule to a linear Volterra integral equation of the first kind, of the form,

$$g(t) = \int_0^t K(t,s)f(s)ds, \quad 0 \le t \le T.$$

Richardson's extrapolation method can also be applied to improve the accuracy of the solution. First kind equations with smooth kernels are required conditions for this routine. Abel type equations should be approximated using the *Abelmid* routine. The approximations of the function f(t) are stored in an array, which is converted into list form so that a plot can be produced by typing *plot(results);*. To execute Richardson's extrapolation, the fifth parameter *R*, must be called and then a plot of the extrapolation can be obtained by typing *print(resultsR);*. Both plots can be shown simultaneously by typing *plot({results,resultsR});*. The solution method for this routine can be seen in chapter five, §5.3.1.

4. Parameters.

Input parameters.

G - The forcing function of the equation. If this function contains an independent variable it must be of the form *t*. G must be of algebraic type, as defined and recognised by Maple. The function should not contain singularities.

K - The kernel of the equation. The independent variables of the equation must be of the form *s* and *t*. K must be of algebraic type, as recognised by Maple. The function should not contain singularities.

h - The stepsize for the approximation to take. This parameter should be of positive real type and be less than one.

n - The number of strips over which the approximation is performed. *n* must be of positive integer type, and greater than four.

R - (optional). The addition of this parameter will execute Richardson's extrapolation method.

Output parameters.

Equation - The entered equation.

Sol - The solution to the equation.

results - The solution is contained in list form and can be plotted by typing *plot(results);*.

resultsR - Richardson's extrapolation solution can be plotted by typing *plot(resultsR);*.

5. Error messages.

ERROR, firstmid expects its first/second argument to be of type algebraic.

The forcing function *G* and the kernel *K* must be of algebraic type. If not then this error message is encountered.

ERROR, firstmid expects its third argument, h, to be of type positive.

The stepsize must be a real positive number.

ERROR, firstmid expects its fourth argument, n, to be of type posint.

The parameter entered for the number of strips, *n*, should be a positive integer, otherwise this error shall occur.

ERROR, since g(0) does not equal zero, a solution does not exist.

For a solution to a first kind equation to exist, the forcing function must equal zero when t=0.

ERROR, stepsize is too large, try less than 1.

Some functions will remain unevaluated if a large stepsize is entered. A stepsize of less than one is recommended.

ERROR, singularity encountered.

An ill-posed problem will not always yield a solution. As a result this error message may appear.

6. Accuracy.

Richardson's extrapolation method is used in the routine to improve the accuracy of the solution. As can be seen in example 2 (section 8, below), although the mid point approximation is the more stable solution, Richardson's extrapolation is closer to the exact solution of f(t)=1. Using the *plot* command is very useful in assessing the behaviour of each solution.

7. Further comments.

Executing Richardson's extrapolation will noticeably slow down the routine. Calculating an extrapolation solution will take approximately twelve times longer when compared to a mid point solution method. This must be considered when choosing the number of strips to be used in an approximation.

8. Examples.

Example 1. Apply the mid point numerical approximation method to the following equation,

$$-\frac{1}{2}\sin\left(\frac{1}{9}t\right)\cos(t) = \frac{9}{4}\int_0^t \cos(t-s)f(s)ds.$$

Below is a listin of inputs and outputs for execution of the problem.

```
> with(volterra):

> firstmid(-1/2*sin(t/9)*cos(t),9/4*cos(t-s),0.01,500);
```

Equation entered: , $-\frac{1}{2}\sin\left(\frac{1}{9}t\right)\cos(t)=\frac{9}{4}\int_0^t\cos(t-s)f(s)ds.$

Mid point solution: $f(t)=-0.01064744965$

Example 2.

Use Richardson's extrapolation method to obtain a solution to,

$$1-\cos(t)=\int_0^t\sin(t-s)f(s)ds.$$

Listing of inputs and outputs.

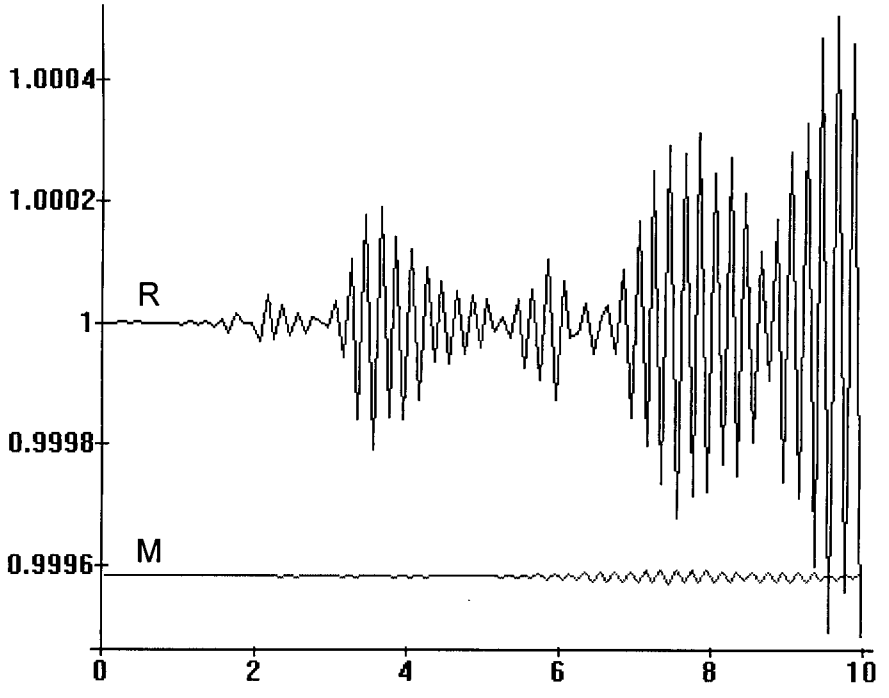
```
> with(volterra):

> firstmid(1-cos(t),sin(t-s),0.1,100,R);
```

Equation entered: , $-\frac{1}{2}\sin\left(\frac{1}{9}t\right)\cos(t)=\frac{9}{4}\int_0^t\cos(t-s)f(s)ds.$

Mid point solution:- $f(t)=0.9995872426$

Richardson's extrapolation:- $f(t)=0.9994828537$

```
> plot({results,resultsR});
```



R=Richardson's extrapolation.
M=Mid point rule approximation.

9 Program listing.

#Mid-point approximation of Volterra first order linear.

```
restart;
firstmid:=proc(G:algebraic,K:algebraic,h:positive,n:posint)
local Func,EXFunc,ExtrapValue,i,j,numapprox,m;
#Check inputs.
if not (eval(subs(t=0,G)))=0 then
ERROR(`Sinceg(0) does not equal zero, a solution does not exist.`);
fi;
if h=1 or h>1 then
ERROR(`step size is too large, try less than 1.`)
fi;
Equation:=Int(K*f(s),s=0..t);
print(`Equation entered:-    `. G=Equation);
#Initialise arrays.
Func:=array(1..n,0..1);
EXFunc:=array(1..(n*3));
ExtrapValue:=array(1..n,0..1);
Func[1,1]:=eval(subs(t=h,G))/(h*eval(subs(t=h,s=h/2,K)));
#Approximation of the integral.
for i from 2 to n do
numapprox:=0;
for j from 0 to i-1 do
numapprox:=numapprox+h*eval(subs(t=i*h,s=(j+1/2)*h,K))*Func[j+1,1];
od;
numapprox:=numapprox=eval(subs(t=i*h,G));
Func[i,1]:=solve(numapprox,Func[i,1]);
#Check for ill-posed problem.
if Func[i,1]=NULL then
ERROR(`Singularity encountered.`);
fi;
od;
if nargs=5 and args[5]=R then
#Richardson's extrapolation.
#Approximation of the integral with stepsize/3.
EXFunc[1]:=eval(subs(t=h/3,G))/((h/3)*eval(subs(t=h/3,s=h/6,K)));
for i from 2 to n*3 do
numapprox:=0;
for j from 0 to i-1 do
numapprox:=numapprox+(h/3)*eval(subs(t=i*h/3,s=(j+1/2)*h/3,K))*EXFunc[j+1];
od;
numapprox:=numapprox=eval(subs(t=i*h/3,G));
EXFunc[i]:=solve(numapprox,EXFunc[i]);
if EXFunc[i]=NULL then
ERROR(`Singularity encountered.`);
fi;
```

```
od;
#Create array of extrapolated values.
for i from 1 to n do
ExtrapValue[i,1]:=EXFunc[3*i-1]+(1/8)*(EXFunc[3*i-1]-Func[i,1]);
od;
#Convert to list form for plot of solution.
for m from 1 to n do
ExtrapValue[m,0]:=(m-1/2)*h;
Func[m,0]:=(m-1/2)*h;
od;
resultsR:=convert(ExtrapValue,list,list);
results:=convert(Func,list,list);
#Print solution.
print(`Mid-point solution:-  `. f(t)=Func[n,1]);
print(`Richardson's extrapolation:-  `. f(t)=ExtrapValue[n,1]);
else
for m from 1 to n do
Func[m,0]:=(m-1/2)*h;
od;
results:=convert(Func,list,list);
print(`Approximate solution:-  `. f(t)=Func[n,1]);
fi;
end;
```

'Abelmid' - Maple routine.

1. Purpose.

Abelmid approximates a linear Volterra integral equation of the first kind of Abel type using the mid-point rule. Aitken's extrapolation method can also be implemented.

2. Specification.

```
with(volterra):
Abelmid(G,K,h,n,A);
```

Algebraic expressions:- G, K.
Real positive constants:- h.
Positive integers:- n.
Optional string expression:- A.

3. Description.

Abelmid will apply the mid-point numerical approximation rule to a linear Volterra integral equation of the first kind of Abel type, of the form,

$$g(t) = \int_0^t \frac{K(t,s)}{(t-s)^u} f(s)ds, \quad 0 \le t \le T,$$

where $K(t,s)$ is smooth, $K(t,t) \neq 0$, and $0 < u < 1$. The process of product integration is used in the application of the mid-point rule. Richardson's extrapolation method can also be applied to Abel type equations, but to further improve the accuracy of the solution Aitken's error correction term is utilised. The approximations of the function f(t) are stored in an array, which is converted into list form so that a plot can be produced by typing *plot(results);*. To execute Aitken's error correction the fifth parameter *A* must be called. A plot of this extrapolation can be shown by typing *print(resultsA);*. Both plots can be shown simultaneously by typing *plot({results,resultsA});*. The solution method of this routine can be seen in chapter five, §5.3.2.

4. Parameters.

Input parameters.

G - The forcing function of the equation. If this function contains an independent variable it must be of the form *t*. G must be of algebraic type, as defined and recognised by Maple. The function should not contain singularities.

K - The kernel of the equation. The independent variables of the equation must be of the form *s* and *t*. K must be of algebraic type, as recognised by Maple. The function should be entered as $K/(t-s)^u$, where K is smooth.

h - The stepsize for the approximation to take. This parameter should be of positive real type and be less than one.

n - The number of strips over which the approximation is performed. *n* must be of positive integer type..

A - (optional). The addition of this parameter will invoke Aitken's error correction method.

Output parameters.

Equation - The entered equation.

Sol - The mid point solution to the equation.

SolA - Aitken's error correction extrapolation solution.

results - The solution is contained in a list form and can be plotted by typing *plot(results);*.

resultsA - Aitken's error correction solution can be plotted by typing *plot(resultsA);*.

5. Error messages.

ERROR, Abelmid expects its first/second argument to be of type algebraic.

The forcing function G and the kernel K must be of algebraic type. If not then this error message is encountered.

ERROR, Abelmid expects its third argument, h, to be of type positive.

The stepsize must be a real positive number.

ERROR, Abelmid expects its fourth argument, n, to be of type posint.

The parameter entered for the number of strips, n, should be a positive integer, otherwise this error will occur.

ERROR, since g(0) does not equal zero a solution does not exist.

For a solution to a first kind equation to exist, the forcing function must equal zero when t=0.

ERROR, stepsize is too large, try less than 1.

Some functions will remain unevaluated if a large stepsize is entered. A stepsize of less than one is recommended.

ERROR, singularity encountered.

An ill-posed problem will not always yield a solution. As a result this error message appears.

6. Accuracy.

Aitken's extrapolation method is used in the routine to improve the accuracy of the solution. The observed error of the solution is approximately $g(t) = \int_0^t \frac{K(t,s)}{(t-s)^u} f(s)ds, \quad 0 \le t \le T,$.

7. Further comments.

Calling Aitken's error correction extrapolation will noticeably slow down the routine. Calculating an extrapolation solution will be very time consuming when compared to the mid-point solution method alone. This must be considered when choosing the number of strips to be used in an approximation.

8. Examples.

Example 1. Apply the mid-point numerical approximation method to the following Abel type equation using Aitken's error correction extrapolation method.

$$\frac{2t}{\sqrt{t+1}} \tanh^{-1}\left(\sqrt{\frac{t}{t+1}}\right) + 2\sqrt{t} = \int_0^t \frac{(1+t+s)}{\sqrt{t-s}} f(s)ds.$$

Listing of inputs and outputs for execution of problem.

```
> with(volterra):

> Abelmid(2*t/sqrt(t+1)*arctan(sqrt(t/(t+1))+2*sqrt(t),(1+t+s)/(sqrt(t-s),0.1,20,A);
```

$$\textit{Equation entered: }, \; 2\frac{t \arctan h\left(\frac{\sqrt{t}}{\sqrt{t+1}}\right)}{\sqrt{t+1}} + 2\sqrt{t} = \int_0^t \frac{(1+t+s)f(s)}{\sqrt{t-s}}\,ds$$

$$\textit{Mid point solution:} \quad f(t) = 0.3389942033$$
$$\textit{Aitken's extrapolation:} \quad f(t) = 0.3389828957$$

Comparing the solutions given here, the mid-point method with Aitken's error correction extrapolation have a convergence order of approximately $O(h^2)$.

9 Program listing.

```
#Mid-point approximation of Volterra first order linear of Abel type.

restart;
Abelmid:=proc(G:algebraic,K:algebraic,h:positive,n:posint)
local Func,EXFunc,EX2Func,ExtrapValue,i,log_value,weight,weight_function,
j,numapprox,m,Denom,Numer,Equation,Sol,SolA;
#Check inputs.
if not (eval(subs(t=0,G)))=0 then
ERROR(`Sinceg(0) does not equal 0, a solution does not exist.`);
fi;
if h=1 or h>1 then
ERROR(`step size is too large, try less than 1.`)
fi;
Equation:=Int(K*f(s),s=0..t);
print(`Equation entered:-    `. G=Equation);
#Seperate kernel.
Denom:=denom(K);
Numer:=numer(K);
Func:=array(1..n,0..1);
EXFunc:=array(1..(n*3));
EX2Func:=array(1..(n*9));
ExtrapValue:=array(1..n,0..1);
#Approximation of the integral.
for i from 1 to n do
numapprox:=0;
for j from 0 to i-1 do
weight_function:=eval(subs(t=i*h,1/Denom));
weight:=Int(weight_function,s=j*h..(j+1)*h);
```

```
numapprox:=numapprox+weight*eval(subs(t=i*h,s=(j+1/2)*h,Numer))*Func[j+1,1];
od;
numapprox:=numapprox=eval(subs(t=i*h,G));
Func[i,1]:=solve(numapprox,Func[i,1]);
Func[i,1]:=evalf(Func[i,1]);
#Check for singularities.
if Func[i,1]=NULL then
ERROR(`Singularity encountered.`);
fi;
od;
Sol:=Func[n,1];
#Aitken's error correction extrapolation.
if nargs=5 and args[5]=A then
#Approximation of the integral with stepsize/3.
for i from 1 to n*3 do
numapprox:=0;
for j from 0 to i-1 do
weight_function:=eval(subs(t=i*(h/3),1/Denom));
weight:=Int(weight_function,s=j*(h/3)..(j+1)*(h/3));
numapprox:=numapprox+weight*eval(subs(t=i*(h/3),s=(j+1/2)*(h/3),Numer))*EXFu
nc[j+1];
od;
numapprox:=numapprox=eval(subs(t=i*h/3,G));
EXFunc[i]:=solve(numapprox,EXFunc[i]);
EXFunc[i]:=evalf(EXFunc[i]);
if EXFunc[i]=NULL then
ERROR(`Singularity encountered.`);
fi;
od;
#Approximation of the integral with stepsize/9.
for i from 1 to n*9 do
numapprox:=0;
for j from 0 to i-1 do
weight_function:=eval(subs(t=i*h/9,1/Denom));
weight:=Int(weight_function,s=j*h/9..(j+1)*h/9);
numapprox:=numapprox+weight*eval(subs(t=i*h/9,s=(j+1/2)*h/9,Numer))*EX2Func
[j+1];
od;
numapprox:=numapprox=eval(subs(t=i*h/9,G));
EX2Func[i]:=solve(numapprox,EX2Func[i]);
EXFunc2[i]:=evalf(EXFunc2[i]);
if EX2Func[i]=NULL then
ERROR(`Singularity encountered.`);
fi;
od;
#Create array of extrapolated values.
for i from 1 to n do
log_value:=log[3]((EXFunc[i*3-1]-Func[i,1])/(EX2Func[i*9-4]-EXFunc[i*3-1]));
```

```
ExtrapValue[i,1]:=(3^log_value*EX2Func[i*9-4]-EXFunc[i*3-1])/(3^log_value-1);
ExtrapValue[i,1]:=evalf(ExtrapValue[i,1]);
od;
#Convert to list form for plot of solution.
for m from 1 to n do
ExtrapValue[m,0]:=(m-1/2)*h;
Func[m,0]:=(m-1/2)*h;
od;
SolA:=ExtrapValue[n,1];
resultsA:=convert(ExtrapValue,list,list);
results:=convert(Func,list,list);
#Print results.
print(`Mid-point solution:-  `. f(t)=Sol);
print(`Aitken's extrapolation:-  `. f(t)=SolA);
else
for m from 1 to n do
Func[m,0]:=(m-1/2)*h;
od;
results:=convert(Func,list,list);
print(`Mid point solution:-  `. f(t)=Sol);
fi;
end;
```

Appendix Two.

Fredholm Maple package description.

The following section provides a documentation of the routines available in the Fredholm package. To run a routine the command
> with(fredholm);
must be entered, after which any of the following can be used;

- **fredcode** - produces a solution to a linear Fredholm integral equation of the second kind by means of a conversion to its ordinary differential form.
- **fredtrap** - trapezium numerical solution method for linear Fredholm integral equations of the second kind.
- **fredsimp** - as above but using Simpson's rule over an even number of intervals.
- **fsimp2** - Numerical solution method using Simpson's 3/8ths rule over an odd number of intervals.
- **fredgreg** - Gregory's fourth order numerical approximation.
- **fredneum** - Neumann series approximation to a linear Fredholm integral equation of the second kind.

Each routine contains its own help page (within Maple) and can be accessed by typing the following command,

> ?*progman_name*;

An example of a help page can be seen in appendix three.

'fredcode' - Maple routine.

1. Purpose.

fredcode produces a solution to a linear Fredholm equation of the second kind by means of a conversion to its ordinary differential form.

2. Specification.

```
with(fredholm):
fredcode(G,K,a,b);
```

Algebraic expressions:- G, K.
Real numbers:- a,b.

3. Description.

fred*code* will convert a linear Fredholm equation of the second kind of the form,

$$f(t) = g(t) + \alpha \int_a^b K(t,s) f(s) ds, \quad -\infty < a < b < \infty,$$

into an ordinary differential equation. It is assumed that the functions involved are sufficiently smooth and well behaved. Some equations containing singularities or an ill-posed problem can be converted to their ordinary differential form, but a solution of this form may not be attainable. The routine will differentiate the equation until an appropriate substitution for the integral term is found. A maximum number of ten derivatives of the original equation will be examined to find a substitution. If an equivalent ordinary differential equation form is found, an expression for f(t) is obtained using Maple's *dsolve* command and an array is set up containing the computed differential equations in order for a series of simultaneous equations to be established. These simultaneous equations are then used to find the values of the constants acquired from the *dsolve* command. The theory behind this method can be found in chapter four, section 4.2.

4. Parameters.

Input parameters.

G - The forcing function of the equation. If this function contains an independent variable it must be of the form *t*. *G* must be of algebraic type, as defined and recognised by Maple.

K - The kernel of the equation. The independent variables of the equation must be of the form *s* and *t*. *K* must be of algebraic type, as recognised by Maple.

a - The lower bound of the integral. *a* must be of *realcons* type, i.e. a real constant.

b - The upper bound of the integral. *b* must also be of *realcons* type.

Output parameters.

ODE1 - The entered equation.

ODE - The equivalent ordinary differential equation.

Constant_Solution1 - The solution for f(t) attained from *ODE* after executing Maple's *dsovle* command.

consprint - The values calculated for the constants in *Constant_Solution1*, (if any exist).

5. Error messages.

ERROR, fredcode expects its first/second argument to be of type algebraic.

The forcing function G and the kernel K must be of algebraic type. If not then this error message is given.

ERROR, fredcode expects its third/fourth argument to be of type realcons.

The bounds of the integral must be entered as real numbers.

ERROR, bounds on the integral are erroneous.

The lower bound must be less than the upper bound.

ERROR, there is no equivalent ODE up to tenth order.

The routine could not find a suitable substitution of the integral term, thus an equivalent ordinary differential equation could not be found. Note that a kernel of the form t^n, where $n>9$, will have an equivalent ordinary differential equation, but since the routine performs a maximum number of ten differentiations such a function will not be recognised.

ERROR, A solution of the ODE for f(t) could not be found.

When an ordinary differential equation for the problem is found, the routine uses the *dsolve* command to obtain an expression for f(t). If there is no solution to the ODE for f(t), this error will occur.

ERROR, A suitable expression for f(t) could not be found.

Similar to the above error except the expression for f(t) found using Maple's *dsolve* command contains integrals which are inappropriate for further computations.

ERROR, Singularity encountered.

This error message occurs when the routine is establishing the constant values for the ODE. The routine finds values for f(a), the derivative of f(a) and so on. If, for instance, the forcing function G is 1/t, such an error message will occur. A singularity in the equation or resultant computations is usually the main cause for this error message.

ERROR, unable to establish constants.

Maple cannot sufficiently reduce a simultaneous equation into a form whereby it can be used by the *solve* command.

Values for the constants cannot be found.

Although not strictly an error message, in certain simultaneous equations values for the constants can be unobtainable. Changing the values for the bounds on the integral may produce values, however.

6. Accuracy.

There are no accuracy measurements.

7. Further comments.

Fundamentally, this routine does the same as the *code* routine except instead of the user having to type the *dsolve* command and find an expression for f(t), the *fredcode* routine performs this task. The reason for this is that Maple's *dsolve* command will not accept derivative variables when entering the initial values, as is required for Volterra equations. However, with Fredholm equations there are no initial

values to be entered, as the constants are calculated later and thus the *dsolve* command is all-inclusive.

8. Examples.

Example 1. In this first example the integral term cannot be substituted after differentiation. Instead it will eventually disappear.

Solve the linear Fredholm equation of the second kind,

$$f(t) = 1 + \frac{1}{2} t \int_{-1}^{1} s^2 t^3 f(s) ds.$$

Below is a listing of inputs and outputs for execution of the problem.

```
> with(fredholm):

> fredcode(1+t/2,s^2*t^3,-1,1);
```

Equation entered: , $f(t) = 1 + \frac{1}{2} t \int_{-1}^{1} s^2 t^3 f(s) ds.$

Equivalent ordinary differential equation: , $\frac{\partial^4}{\partial t^4} f(t) = 0$

Solution for f(t): , $f(t) = \_C1 + \_C2t + \_C3t^2 + \_C4t^3$

Values for constants: ,

$$\{\_C4 = 0.6666666670, \_C3 = 0.5000000003\,10^{-9}, \_C1 = 1.00000000, \_C2 = 0.5000000010\}$$

Substituting these values into the function f(t), the solution is approximately,

$$f(t) = 1 + \frac{1}{2} t + \frac{2}{3} t^3.$$

Example 2.

In this example the kernel is of convolution type and the integral term can be substituted after two differentiations. Solve

$$f(t) = \frac{1}{\cos(t)} + \frac{1}{2} \int_{-2}^{1} \sin(t-s) f(s) ds.$$

Listing of inputs and outputs for execution of problem.

```
> with(fredholm):

> fredcode(1/cos(t),1/2*sin(t-s),-2,1);
```

Equation entered: , $f(t) = \frac{1}{\cos(t)} + \frac{1}{2} \int_{-2}^{1} \sin(t-s) f(s) ds.$

Equivalent ordinary differential equation: , $\frac{\partial^2}{\partial t^2} f(t) = 2\frac{\sin(t)^2}{\cos(t)^3} + 2\frac{1}{\cos(t)} - f(t).$

Solution for f(t): , $f(t) = \frac{1 - 2\cos(t)^2 + \_C1 \sin(t)\cos(t) + \_C2\cos(t)^2}{\cos(t)}$

Values for constants:

$$,\{\_C1 = 0.9965485831 - 0.7737616886I, \_C2 = 1.383837890 - 1.035979630I\}.$$

The solution for this example contains the letter I, defined by Maple as a complex number, where $I = \sqrt{-1}$.

Example 3.

The final example for this routine has a kernel of centro-symmetric form. That is K(t,s) = K(a + b – t, a + b – s). Solve

$$f(t) = \sqrt{t+1} + \int_0^1 (1-t)^2 \sqrt{(1-s)} f(s) ds.$$

Listing of inputs and outputs.

```
> with(fredholm):

> fredcode(sqrt(t+1),(1-t)^2*sqrt(1-s),0,1);
```

Equation entered: , $f(t) = \sqrt{t+1} + \int_0^1 (1-t)^2 \sqrt{(1-s)} f(s) ds.$

Equivalent ordinary differential equation: , $\frac{\partial^3}{\partial t^3} f(t) = \frac{3}{8} \frac{1}{(t+1)^{5/2}}.$

Solution for f(t): , $f(t) = \frac{\_C3t^2\sqrt{t+1} + \_C2t\sqrt{t+1} + t + 1 + \_C1\sqrt{t+1}}{\sqrt{t+1}}$

Values for constants: $\{\_C1 = 1.099557430, \_C3 = 1.099557429, \_C2 = -2.199114859\}$

9 Program listing.

```
#Conversion of Fredholm integral equation to ODE form.

restart;
fredcode:=proc(G:algebraic,K:algebraic,lower:realcons,upper:realcons)
local diffeqns,conseqns,simuleqns,cons,stopthis,ODE1,Intsub,ODE,count,EndIntsub,
Constant_Solution1,sing,tempsub,j,solsimul,simulcons,consprint,Constant_Solution;
#Check inputs.
if upper<lower then
ERROR(`bounds on the integral are erroneous.`);
fi;
diffeqns:=array[1..10];
conseqns:=array[1..10];
simuleqns:=array[1..10];
cons:=array[1..10];
stopthis:=no;
diffeqns[1]:=eval(subs(t=lower,G+Int(K*f(s),s=lower..upper)));
ODE1:=f(t)=G+Int(K*f(s),s=lower..upper);
Intsub:=Int(K*f(s),s=lower..upper);
ODE:=diff(ODE1,t);
count:=1;
#loop to establish whether differential equations contain
#the original integral.
while has(integrand(ODE),integrand(Intsub))=false and
```

```
has(integrand(ODE),-integrand(Intsub))=false and
count<10 and stopthis=no do
if integrand(ODE)={} then stopthis:=yo else
count:=count+1;
stopthis:=no;
sing:=traperror(eval(subs(t=lower,rhs(ODE))));
if sing=`division by zero` then
ERROR(`Singularity encountered.`);
fi;
diffeqns[count]:=eval(subs(t=lower,rhs(ODE)));
ODE:=diff(ODE,t);
fi;
od;
if stopthis=no then
print(`Equation entered:  `,ODE1);
#Find substitution.
if not (integrand(Intsub))=0 then
EndIntsub:=(integrand(ODE)/integrand(Intsub))*(f(t)-G);
else EndIntsub:=f(t)-G;
fi;
ODE:=subs(Int(integrand(ODE),s=lower..upper)=EndIntsub,ODE);
print(`Equivalent ordinary differential equation:  `,ODE);
go:=yo;
elif stopthis=yo then print(`Equation entered:  `,ODE1);
print(`Equivalent ordinary differential equation:  `,ODE);
go:=yo;
else lprint(`There is no equivalent ordinary differential equation, (up to 10th order).`);
go:=no;
fi;
if not go=no then
# Find dsolve solution.
Constant_Solution1:=dsolve(ODE,f(t));
if Constant_Solution1=NULL then
ERROR(`A solution of the ODE for f(t) could not be found.`);
elif has(Constant_Solution1,int)=true then
ERROR(`A suitable expression for f(t) could not be found.`);
fi;
Constant_Solution:=Constant_Solution1;
tempsub:=rhs(eval(subs(t=s,Constant_Solution1)));
for j from 1 to count do
sing:=traperror(eval(subs(t=lower,rhs(Constant_Solution))));
if sing=`division by zero` then
ERROR(`Singularity encountered.`);
fi;
conseqns[j]:=evalf(value(eval(subs(t=lower,rhs(Constant_Solution)))));
Constant_Solution:=diff(Constant_Solution,t);
sing:=traperror(evalf(value(eval(subs(f(s)=tempsub,diffeqns[j])))));
if sing=lasterror then
```

```
ERROR(`Unable to establish constants.`);
fi;
simuleqns[j]:=evalf(value(eval(subs(f(s)=tempsub,diffeqns[j]))))=conseqns[j];
cons[j]:=_C.j;
od;
#Find constants for f(t).
solsimul:=convert(simuleqns,set);
simulcons:=convert(cons,set);
consprint:=solve(solsimul,simulcons);
print(`Solution for f(t):  `,Constant_Solution1);
print(``);
if consprint=NULL then
print(`Values for the constants cannot be found, try a larger interval.`)
else
print(`Values for constants:  `,consprint);
fi;
fi;
end;
```

10. Programming problems and effectiveness of Maple.

The part of the program in which the integral term is eliminated from the problem is essentially the same as for the *code* routine and hence the limitations of Maple in this area are the same (see *code* documentation, section 10). A significant drawback for this routine occurs in solving the numbers of simultaneous equations when finding values for the constants. Complex solutions for f(t) attained using the *dsolve* command can create problems when substituting f(t) into the integral terms to obtain a series of simultaneous equations (see chapter four, §4.2, for the specific solution method). This is due to Maple's inefficiency in evaluating integrals containing many terms.

'fredtrap' - Maple routine.

1. Purpose.

fredtrap applies the trapezium numerical approximation method to a linear Fredholm equation of the second kind.

2. Specification.

with(fredholm):
volttrap(G,K,a,b,h);

Algebraic expressions:- G, K.
Real numbers:- a,b.
Real positive constants:- h.

3. Description.

fredtrap will apply the trapezium numerical approximation to a linear Volterra equation of the second kind of the form,

$$f(t) = g(t) + \int_a^b K(t,s)f(s)ds, \quad -\infty < a < b < \infty.$$

The number of strips required for the routine are determined by the stepsize and the bounds on the integral. Each approximate value is stored in a two dimensional array which is converted to list form and can then be used in conjunction with Maple's *plot* command to produce a graph of the solution. The solution method of this routine can be seen in chapter five, §5.4.

4. Parameters.

Input parameters.

G - The forcing function of the equation. If this function contains an independent variable it must be of the form *t*. G must be of algebraic type, as defined and recognised by Maple. *G* and *K* should not contain singularities.

K - The kernel of the equation. The independent variables of the equation must be of the form *s* and *t*. K must be of algebraic type, as recognised by Maple.

a - The lower bound of the integral. *a* must be of *realcons* type. i.e. a real constant.

b - The upper bound of the integral. *b* must also be of *realcons* type.

h - The stepsize for the approximation to take. This parameter should be of positive real type and be less than one.

Output parameters.

Equation - The entered equation.

Sol - The solution to the equation.

results - The solution is contained in a list form and can be plotted by typing *plot(results);*.

5. Error messages.

ERROR, fredtrap expects its first/second argument to be of type algebraic.

The forcing function *G* and the kernel *K* must be of algebraic type. If not then this error message is encountered.

ERROR, fredtrap expects its third/fourth argument, a/b, to be of type realcons.
The parameters given for the bounds of the integral, should be real numbers.
ERROR, fredtrap expects its fifth argument, h, to be of type positive.
The stepsize parameter should be a positive real constant.
ERROR, Stepsize is not divisible into bounds of integral.
To calculate the number of strips required for the approximation, the routine evaluates $n = (b-a)/h$. The value *n* should be an integer, otherwise this error will occur.
ERROR, stepsize is too large, try less than 1.
Some functions will remain unevaluated if a large stepsize is entered. A stepsize of less than one is recommended.
ERROR, (in fredtrap) division by zero.
When the problem contains singularities this error message will occur.

6. Accuracy.

Using the result obtained in example 1, below, and comparing it with the exact solution of $f(t) = 1 + \frac{1}{2}t + \frac{2}{3}t^3$, when t=1, the *fredtrap* routine has an observed error of 8.33×10^{-4}, which is an acceptable error for the trapezium solution method.

7. Further comments.

The approximations are stored in a two dimensional array, one for the numerical approximation and one for the value of *t*. Once the approximation is complete the array is converted into list form so it can be fed into the plot command. When using the *plot* command a plot of f(t) will be given against *t*. Floating point arithmetic is used in the computations. As the routine method is based on the solution of simultaneous equations, a large computed value of *n* will dramatically slow down the routine.

8. Examples.

Example 1. Taking example 1 from the *fredcode* documentation and applying the trapezium rule,

$$f(t) = 1 + \frac{1}{2}t + \int_{-1}^{1} s^2 t^3 f(s) ds.$$

the listings of inputs and outputs obtained from the routine are

```
> with(fredholm):

> fredtrap(1+t/2,s^2*t^3,-1,1,0.05);
```

Equation entered: , $f(t) = 1 + \frac{1}{2}t + \int_{-1}^{1} s^2 t^3 f(s) ds.$

Approximate solution: f(t)= 2.167500002.

9 Program listing.

 # Trapezium approximation of Fredholm second order linear.

```
restart;
fredtrap:=proc(G:algebraic,K:algebraic,lower:realcons,upper:realcons,h:positive)
local intervals,Func,i,middle_part,j,eq,solsimul,conssimul,Equation,Sol;
intervals:=convert((upper-lower)/h,rational);
#Check inputs.
if type(intervals,fraction)=true then
ERROR(`Stepsize is not divisible into bounds of integral.`)
fi;
if upper<lower then
ERROR(`Bounds on the integral are erroneous.`);
fi;
if not h<1 then
ERROR(`Stepsize is too large, try less than 1.`);
fi;
Func:=array(0..intervals);
Cons:=array(0..intervals);
results:=array(0..intervals,0..1);
for i from 0 to intervals do
Cons[i]:=_C.i;
od;
middle_part:=0;
#Trapezium approximation.
for j from 1 to intervals-1 do
middle_part:=middle_part+evalf(eval(subs(s=lower+j*h,K)))*Cons[j];
od;
eq:=G+(1/2)*h*(evalf(eval(subs(s=lower,K)))*Cons[0]+evalf(eval(subs(s=upper,K)))
*Cons[intervals])+h*middle_part;
for i from 0 to intervals do
Func[i]:=Cons[i]=evalf(eval(subs(t=lower+i*h,eq)));
od;
solsimul:=convert(Func,set);
conssimul:=convert(Cons,set);
answer:=solve(solsimul,conssimul);
assign(answer);
Sol:=_C.intervals;
for i from 0 to intervals do
results[i,1]:=_C.i;
results[i,0]:=lower+i*h;
_C.i:=evaln(_C.i);
od;
results:=convert(results,list,list);
Equation:=G+Int(K*f(s),s=lower..upper);
#Print results.
print(`Equation entered:-    `. f(t)=Equation);
print(`Approximate solution:-  `. f(t)=Sol);
end;
```

'fredsimp' - Maple routine.

1. Purpose.

fredsimp is a numerical approximation for linear Fredholm equations of the second kind using the repeated Simpson's rule over an even number of intervals

2. Specification.

```
with(Fredholm):
fredsimp(G,K,a,b,h);
```

Algebraic expressions:- G, K.
Real numbers:- a,b.
Real positive constants:- h.

3. Description.

fredsimp will apply the repeated Simpson's numerical approximation rule to a linear Fredholm equation of the second kind of the form,

$$f(t) = g(t) + \int_a^b K(t,s)f(s)ds, \quad -\infty < a < b < \infty.$$

The number of strips to be used is calculated by the routine from the values given for the stepsize and the bounds on the integral. Only if the value of n (the number of strips) is even will the routine continue, otherwise it will terminate. Each approximate value is stored in a two dimensional array, which is converted to list form and can then be used in conjunction with Maple's *plot* command to produce a graph of the solution. The solution method of this routine can be seen in chapter five, §5.4.

4. Parameters.

Input parameters.

G - The forcing function of the equation. If this function contains an independent variable it must be of the form *t*. G must be of algebraic type, as defined and recognised by Maple. The function should not contain singularities.

K - The kernel of the equation. The independent variables of the equation must be of the form *s* and *t*. K must be of algebraic type, as recognised by Maple. The function should not contain singularities.

a - The lower bound of the integral. *a* should be of *realcons* type. i.e. a real constant.

b - The upper bound of the integral. *b* should be of *realcons* type as well.

h - The stepsize for the approximation to take. This parameter should be of positive real type and be less than one.

Output parameters.

Equation - The entered equation.

Sol - The solution to the equation.

results - The solution is contained in a list form and can be plotted by typing *plot(results);*.

5. Error messages.

ERROR, fredsimp expects its first/second argument to be of type algebraic.

The forcing function G and the kernel K must be of algebraic type. If not then this error message is encountered.

ERROR, fredsimp expects its third/fourth argument to be of type realcons.

The values given for the bounds of the integral should be real numbers.

ERROR, fredsimp expects its fifth argument, h, to be of type positive.

The stepsize value h should be a real positive number otherwise this error will occur.

ERROR, Bounds on the integral are erroneous.

The lower bound parameter a must be less than the upper bound parameter b.

ERROR, Stepsize is not divisible into bounds of integral.

The value for the number of strips required for the interval must evaluate to an integer.

ERROR, The number of intervals is odd. Use fsimp2 routine.

If the value for the number of strips needed for the approximation is odd the *fsimp2* routine should be used.

ERROR, stepsize is too large, try less than 1.

Some functions will remain unevaluated if a large stepsize is entered. A stepsize of less than one is recommended.

ERROR, (in fredsimp) division by zero.

When the problem contains singularities this error message will occur.

6. Accuracy.

The result obtained from the example below has an error of -1×10^{-9}, which is extremely accurate for this example. The expected observed error is approximately $O(h^5)$.

7. Further comments.

See *fredtrap* documentation, section 7.

8. Examples.

Example 1. Using the equation seen in the *fredtrap* documentation apply Simpson's rule to,

$$f(t)=1+\frac{1}{2}t+\int_{-1}^{1}s^2t^3f(s)ds.$$

Listing of inputs and outputs for execution of problem.

```
> with(fredholm):

> fredsimp(1+t/2,s2*t^3,-1,1,0.1);
```

Equation entered: , $f(t)=1+\frac{1}{2}t+\int_{-1}^{1}s^2t^3f(s)ds.$

Approximate solution: f(t)= 2.166666666.

9 Program listing.

```
#Simpson's even solution method, for Fredholm.
```

```
restart;
fredsimp:=proc(G:algebraic,K:algebraic,lower:realcons,upper:realcons,h:positive)
local
intervals,Func,Consi,middle_part1,middle_part2,eq,solsimul,conssimul,Sol,Equation;
intervals:=convert((upper-lower)/h,rational);
#Check inputs.
if upper<lower then
ERROR(`Bounds on the integral are erroneous.`)
fi;
if type(intervals,fraction)=true then
ERROR(`Stepsize is not divisible into bounds of integral.`)
elif type(intervals,odd)=true then
ERROR(`The number of intervals is odd. Use fsimp2 routine.`);
fi;
if not h<1 then
ERROR(`Stepsize is too large, try less than 1.`);
fi;
Func:=array(0..intervals);
Cons:=array(0..intervals);
results:=array(0..intervals,0..1);
for i from 0 to intervals do
Cons[i]:=_C.i;
od;
#Simpson's approximation.
middle_part1:=0;
middle_part2:=0;
for i from 1 to intervals/2 do
middle_part1:=middle_part1+evalf(subs(s=lower+(2*i-1)*h,K))*Cons[2*i-1];
od;
for i from 1 to (intervals/2)-1 do
middle_part2:=middle_part2+evalf(subs(s=lower+2*i*h,K))*Cons[2*i];
od;
eq:=G+h/3*(evalf(subs(s=lower,K))*Cons[0]+4*middle_part1+2*middle_part2+evalf
(subs(s=upper,K))*Cons[intervals]);
for i from 0 to intervals do
Func[i]:=Cons[i]=evalf(subs(t=lower+i*h,eq));
od;
#Solve simultaneous equations.
solsimul:=convert(Func,set);
conssimul:=convert(Cons,set);
answer:=solve(solsimul,conssimul);
assign(answer);
for i from 0 to intervals do
results[i,1]:=_C.i;
results[i,0]:=lower+i*h;
od;
results:=convert(results,list,list);
```

```
Sol:=_C.intervals;
for i from 0 to intervals do
_C.i:=evaln(_C.i);
od;
Equation :=G+Int(K*f(s),s=lower..upper);
print(`Equation entered:-  `.f(t)=Equation);
print(`Approximate solution:-  `. f(t)=Sol);
end;
```

'fsimp2' - Maple routine.

1. Purpose.

fsimp2 is a numerical approximation for linear Fredholm equations of the second kind using the repeated Simpson's rule over an odd number of intervals

2. Specification.

```
with(Fredholm):
fsimp2(G,K,a,b,h);
```

Algebraic expressions:- G, K.
Real numbers:- a,b.
Real positive constants:- h.

3. Description.

fsimp2 will apply the repeated Simpson's numerical approximation rule to a linear Fredholm equation of the second kind of the form,

$$f(t) = g(t) + \int_a^b K(t,s)f(s)ds, \quad -\infty < a < b < \infty.$$

This routine is essentially the same as the *fredsimp* routine, but with slightly different weights. The number of strips to be used is calculated by the routine from the values given for the stepsize and the bounds on the integral. Only if the value of *n*, (the number of strips) is odd will the routine continue, otherwise it will terminate. Each approximate value is stored in a two dimensional array, which is converted to list form and can then be used in conjunction with Maple's *plot* command to produce a graph of the solution. The solution method of this routine can be seen in chapter five, §5.4.

4. Parameters.

Input parameters.

G - The forcing function of the equation. If this function contains an independent variable it must be of the form *t*. G must be of algebraic type, as defined and recognised by Maple. The function should not contain singularities.

K - The kernel of the equation. The independent variables of the equation must be of the form *s* and *t*. K must be of algebraic type, as recognised by Maple. The function should not contain singularities.

a - The lower bound of the integral. *a* should be of *realcons* type. i.e. a real constant.

b - The upper bound of the integral. *b* should be of *realcons* type as well.

h - The stepsize for the approximation to take. This parameter should be of positive real type and be less than one.

Output parameters.

Equation - The entered equation.

Sol - The solution to the equation.

results - The solution is contained in a list form and can be plotted by typing *plot(results);*.

5. Error messages.

ERROR, fsimp2 expects its first/second argument to be of type algebraic.

The forcing function G and the kernel K must be of algebraic type. If not then this error message is encountered.

ERROR, fsimp2 expects its third/fourth argument to be of type realcons.

The values given for the bounds of the integral should be real numbers.

ERROR, fsimp2 expects its fifth argument, h, to be of type positive.

The stepsize value h should be a real positive number otherwise this error will occur.

ERROR, Bounds on the integral are erroneous.

The lower bound parameter, a, must be less than the upper bound parameter, b.

ERROR, Stepsize is not divisible into bounds of integral.

The value for the number of strips required for the interval must evaluate to an integer.

ERROR, The number of intervals is even. Use fredsimp routine.

If the value for the number of strips needed for the approximation is even the *fredsimp* routine should be used.

ERROR, Number of intervals must be five or more.

The approximation requires a minimum of five strips to execute.

ERROR, stepsize is too large, try less than 1.

Some functions will remain unevaluated if a large stepsize is entered. A stepsize of less than one is recommended.

ERROR, (in fsimp2) division by zero.

When the problem contains singularities this error message will occur.

6. Accuracy.

Once again the example from the *fredtrap* documentation is used to provide a comparison of solution methods. In this routine, when using an odd number of strips and a stepsize close to 0.1 of 2/21, the error is considerable, being -8.55×10^{-3}. When comparing this to the result obtained from the *fredsimp* routine it is clear that the latter routine is by far the more preferable.

7. Further comments.

See *fredtrap* documentation, section 7.

8. Examples.

Example 1. Apply Simpson's rule to,

$$f(t)=1+\frac{1}{2}t+\int_{-1}^{1}s^2t^3f(s)ds.$$

Below is a listing of inputs and outputs for execution of the problem.

```
> with(fredholm):

> fsimp2(1+t/2,s2*t^3,-1,1,2/21);
```

Equation entered: , $f(t)=1+\frac{1}{2}t+\int_{-1}^{1}s^2t^3f(s)ds.$

Approximate solution: f(t)= 2.158113773.

9 Program listing.

```
#Simpson's solution method, for Fredholm.

restart;
fsimp2:=proc(G:algebraic,K:algebraic,lower:realcons,upper:realcons,h:positive)
local intervals,Func,Cons,i,middle_part1,middle_part2,eq,
solsimul,conssimul,Sol,Equation;
intervals:=convert((upper-lower)/h,rational);
#Check inputs.
if upper<lower then
ERROR(`Bounds on the integral are erroneous.`)
fi;
if type(intervals,fraction)=true then
ERROR(`Stepsize is not divisible into bounds of integral.`)
elif type(intervals,even)=true then
ERROR(`The number of intervals is even. Use fredsimp routine.`);
fi;
if intervals<5 then
ERROR(`Number of intervals must be five or more.`);
fi;
if not h<1 then
ERROR(`Stepsize is too large, try less than 1.`);
fi;
Func:=array(0..intervals);
Cons:=array(0..intervals);
results:=array(0..intervals,0..1);
for i from 0 to intervals do
Cons[i]:=_C.i;
od;
#Simpson's rule.
middle_part1:=0;
middle_part2:=0;
for i from 1 to (intervals-3)/2 do
middle_part1:=middle_part1+evalf(eval(subs(s=lower+(2*i-1)*h,K)))*Cons[2*i-1];
od;
for i from 1 to (intervals-5)/2 do
middle_part2:=middle_part2+evalf(eval(subs(s=lower+2*i*h,K)))*Cons[2*i];
od;
eq:=G+h/3*(evalf(eval(subs(s=lower,K)))*Cons[0]+4*middle_part1+2*middle_part2
+(17/8)*evalf(eval(subs(s=upper-3*h,K)))*Cons[intervals-
3]+(27/8)*evalf(eval(subs(s=upper-2*h,K)))*Cons[intervals-
2]+(27/8)*evalf(eval(subs(s=upper-h,K)))*Cons[intervals-
1]+evalf(eval(subs(s=upper,K)))*Cons[intervals]);
for i from 0 to intervals do
Func[i]:=Cons[i]=evalf(eval(subs(t=lower+i*h,eq)));
od;
```

```
#Solve simultaneous equations.
solsimul:=convert(Func,set);
conssimul:=convert(Cons,set);
Sol:=solve(solsimul,conssimul);
assign(Sol);
Sol:=_C.intervals;
for i from 0 to intervals do
results[i,1]:=_C.i;
results[i,0]:=lower+i*h;
_C.i:=evaln(_C.i);
od;
results:=convert(results,list,list);
#Print results.
Equation:=G+Int(K*f(s),s=lower..upper);
print(`Equation entered:-  `.f(t)=Equation);
print(`Approximate solution:-  `. f(t)=Sol);
end;
```

'fredgreg' - Maple routine.

1. Purpose.

fredgreg applies Gregory's fourth order numerical approximation to a linear Fredholm equation of the second kind.

2. Specification.

```
with(fredholm):
fredgreg(G,K,lower,upper,h);
```

Algebraic expressions:- G, K.
Real numbers:- lower, upper.
Real positive constants:- h.

3. Description.

fredgreg approximates a linear Fredholm equation of the second kind, of the form,

$$f(t) = g(t) + \int_a^b K(t,s)f(s)ds, \quad -\infty < a < b < \infty.$$

by the application of Gregory's fourth order approximation rule.

Each approximation is calculated by solving numerous simultaneous equations. Each approximate value is stored in a two dimensional array which is converted to list form and can then be used in conjunction with Maple's *plot* command to produce a graph of the solution. The solution method for this routine can be seen in chapter five, §5.4.

4. Parameters.

Input parameters.

G - The forcing function of the equation. If this function contains an independent variable it must be of the form *t*. G must be of algebraic type, as defined and recognised by Maple. The function should not contain singularities.

K - The kernel of the equation. The independent variables of the equation must be of the form *s* and *t*. K must be of algebraic type, as recognised by Maple. The function should not contain singularities.

lower - The lower bound of the integral. *lower* should be of *realcons* type. i.e. a real constant.

upper - The upper bound of the integral. *upper* should also be of *realcons* type.

h - The stepsize for the approximation to take. This parameter should be of positive real type and be less than one.

Output parameters.

Equation - The entered equation.

Sol - The solution to the equation.

results - The solution is contained in a list form and can be plotted by typing *plot(results);*.

5. Error messages.

ERROR, fredgreg expects its first/second argument to be of type algebraic.

The forcing function G and the kernel K must be of algebraic type. If not then this error message is encountered.

ERROR, fredgreg expects its third/fourth argument to be of type realcons.

The values given for the bounds of the integral should be real numbers.

ERROR, fredgreg expects its fifth argument, h, to be of type positive.

The stepsize value h should be a real positive number, otherwise this error will occur.

ERROR, Bounds on the integral are erroneous.

The lower bound parameter, a, must be less than the upper bound parameter, b.

ERROR, Stepsize is not divisible into bounds of integral.

The value for the number of strips required for the interval must evaluate to an integer.

ERROR, Routine requires a minimum of five strips. Try smaller stepsize.

The approximation requires a minimum of five strips to execute.

ERROR, stepsize is too large, try less than one.

Some functions will remain unevaluated if a large stepsize is entered. A stepsize of less than one is recommended.

ERROR, (in fredgreg) division by zero.

When the problem contains singularities this error message will occur.

6. Accuracy.

Again, the example from the *fredtrap* documentation is used in a comparison of solution methods. The *fredgreg* routine produces an approximate solution (correct to 9 decimal places) of 2.166666667. Generally this solution method has an observed error of approximately $O(h^4)$ and so it should be noted that this example is easily approximated.

7. Further comments.

See *fredtrap* documentation, section 7.

8. Examples.

Example 1. Apply Gregory's rule to,

$$f(t) = 1 + \frac{1}{2}t + \int_{-1}^{1} s^2 t^3 f(s) ds.$$

Listing of inputs and outputs for execution of problem.

```
> with(fredholm):

> fredgreg(1+t/2,s2*t^3,-1,1,0.1);
```

Equation entered: , $f(t) = 1 + \frac{1}{2}t + \int_{-1}^{1} s^2 t^3 f(s) ds.$

Approximate solution: f(t)= 2.166666667.

9 Program listing.

#Gregory approximation, for Fredholm.

```
restart;
fredgreg:=proc(G:algebraic,K:algebraic,lower:realcons,upper:realcons,h:positive)
local intervals,Func,Cons,i,middle_part,eq,solsimsul,conssimul,Sol,Equation;
intervals:=convert((upper-lower)/h,rational);
#Check inputs.
if upper<lower then
ERROR(`Bounds on the integral are erroneous.`);
fi;
if type(intervals,fraction)=true then
ERROR(`Stepsize is not divisible into bounds of integral.`)
fi;
if intervals<5 then
ERROR(`Routine requires a minimum of five strips. Try smaller stepsize.`);
fi;
if not h<1 then
ERROR(`Stepsize is too large, Try less than one.`);
fi;
Func:=array(0..intervals);
Cons:=array(0..intervals);
results:=array(0..intervals,0..1);
for i from 0 to intervals do
Cons[i]:=_C.i;
od;
#Gregory's approximation.
middle_part:=0;
for i  from 3 to intervals-3 do
middle_part:=middle_part+evalf(eval(subs(s=lower+i*h,K)))*Cons[i];
od;
eq:=G+h*((3/8)*evalf(eval(subs(s=lower,K)))*Cons[0]+(7/6)*evalf(eval(subs(s=lowe
r+h,K)))*Cons[1]+(23/24)*evalf(eval(subs(s=lower+2*h,K)))*Cons[2]+middle_part+
(23/24)*evalf(eval(subs(s=upper-2*h,K)))*Cons[intervals-
2]+(7/6)*evalf(eval(subs(s=upper-h,K)))*Cons[intervals-
1]+(3/8)*evalf(eval(subs(s=upper,K)))*Cons[intervals]);
for i from 0 to intervals do
Func[i]:=Cons[i]=evalf(eval(subs(t=lower+i*h,eq)));
od;
#Solve simultaneous equations.
solsimul:=convert(Func,set);
conssimul:=convert(Cons,set);
Sol:=solve(solsimul,conssimul);
assign(Sol);
Sol:=_C.intervals;
for i from 0 to intervals do
results[i,1]:=_C.i;
```

```
results[i,0]:=lower+i*h;
_C.i:=evaln(_C.i);
od;
results:=convert(results,list,list);
#Print results.
Equation:=G+Int(K*f(s),s=lower..upper);
print(`Equation entered:-  `.f(t)=Equation);
print(`Approximate solution:-  `.f(t)=Sol);
end;
```

'fredneum' - Maple routine.

1. Purpose.

fredneum produces a solution to a linear Fredholm integral equation of the second kind by applying the Neumann series iterative method.

2. Specification.

with(fredholm):
fredneum(G,K,lower,upper,iter);

Algebraic expressions:- G, K.
Real numbers:- lower, upper.
Real positive constants:- iter (optional).

3. Description.

The routine *fredneum* applies the Neumann iterative approximation series to a linear Fredholm integral equation of the second kind of the form,

$$f(t) = g(t) + \int_a^b K(t,s)f(s)ds, \quad -\infty < a < b < \infty.$$

to produce a solution for f(t).

The integral of each iterative step is evaluated using Maple's *value* command and, providing an explicit expression can be obtained, a solution is be given for f(t). The solution for f(t) will then be compared with the solution for $f(t_{n-1})$ to determine whether the iterative series is convergent. The formula for this process is,

$$\left|f(t_n) - f(t_{n-1})\right| < \frac{1}{(b-a)},$$

where the functions $f(t_n)$ and $f(t_{n-1})$ are evaluated for t = b.

The number of iterations performed by the routine can be assigned in the parameter inputs, but if no parameter is entered a default value of ten is applied. The solution method behind this routine can be seen in chapter five, §5.1.

4. Parameters.

Input parameters.

G - The forcing function of the equation. If this function contains an independent variable it must be of the form *t*. G must be of algebraic type, as defined and recognised by Maple. The function should not contain singularities.

K - The kernel of the equation. The independent variables of the equation must be of the form *s* and *t*. K must be of algebraic type, as recognised by Maple. The function should not contain singularities.

lower - The lower bound of the integral. *lower* should be of *realcons* type. i.e. a real constant.

upper - The upper bound of the integral. *upper* should also be of *realcons* type.

iter - (optional). The number of iterations used in the Neumann series approximation. If no fifth parameter is entered a default value of ten iterations will be used.

Output parameters.

Equation - The entered equation.

Sol - The solution to the equation. In this routine the output parameter is defined as global so that it can be used with Maple's *plot* command after execution of the routine. It is often more useful to analyse a graph with this solution method as some results can be complex.

5. Error messages.

ERROR, fredneum expects its first/second argument to be of type algebraic.

The forcing function G and the kernel K must be of algebraic type. If not then this error message is encountered.

ERROR, fredneum expects its third/fourth argument to be of type realcons.

The values given for the bounds of the integral should be real numbers.

ERROR, bounds on the integral are erroneous.

The parameter *lower* must be given as less than the value for *upper*.

ERROR, variable for number of iterations must be of type (posint).

The optional parameter of *iter* defining the desired number of iterations must be a positive integer.

ERROR, Number of iterations must be three or more.

To obtain enough data to establish convergence, a minimum number of three iterations must be performed.

ERROR, (in fredneum) division by zero.

When the problem contains singularities this error message will occur.

ERROR, Unable to evaluate integral.

Maple's built in *value* command (used to solve the integral term of the iteration) cannot sufficiently reduce the integrand to exclude the integral sign.

ERROR, (in fredneum) cannot evaluate Boolean.

When checking for convergence the routine compares the difference between the last two iterations against the inverse of the upper bound minus the lower bound (see section three). If Maple cannot sufficiently evaluate this condition, this error will be returned.

6. Accuracy.

The routine performs a check on the last two iterations to verify whether the computed solution is convergent.

7. Further comments.

The Neumann iterative approximation series routine works impeccably well with simple equations, producing near exact solutions after only a small number of iterations (see below). However, when the kernel function becomes a little more complex, the evaluation of the integral can become restricted and the solution can be very lengthy. In such circumstances using the command *plot(Sol,t=lower..upper);* (where *lower* and *upper* are the parameters given for the bounds on the integral) can present a clearer graphical interpretation of the solutions behaviour.

8. Examples.

Example 1. Apply Neumann's iterative approximation series to the equation,

$$f(t) = 1 + \frac{1}{2}t + \int_{-1}^{1} s^2 t^3 f(s)\,ds.$$

This problem has been posed for the previous numerical solution methods and is used here to show how efficient this method can be. Although the default value of ten iterations is used in this example, the exact solution is reached after just three iterations.

Listing of inputs and outputs.

```
> with(fredholm):

> fredneum(1+t/2,s2*t^3,-1,1);
```

$$\textit{Equation entered: , } f(t) = 1 + \frac{1}{2}t + \int_{-1}^{1} s^2 t^3 f(s)\,ds.$$

$$\textit{Solution of equation: } 1 + \frac{1}{2}t + \frac{2}{3}t^3.$$

Neumann series approximation appears to converge.

Example 2. This example shows how the Neumann series can produce a lengthy solution to a simple problem. The minimum number of three iterations is used to restrict the size of the output. Solve

$$f(t) = t + \int_{-1}^{1} \cos(t-s) f(s)\,ds.$$

Listings of inputs and outputs.

```
> with(fredholm):

> fredneum(t,cos(t-s),-1,1,3);
```

$$\textit{Equation entered: , } f(t) = t + \int_{-1}^{1} \cos(t-s) f(s)\,ds.$$

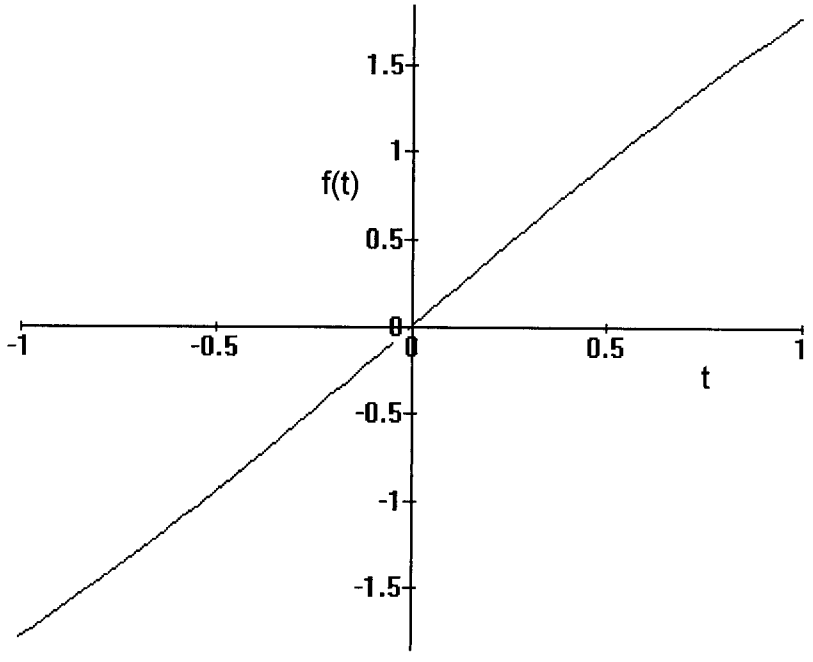
Solution of equation:

$$\begin{aligned} f(t) = t &+ 2.25000000\cos(t-1) - 2.25000000\sin(t-1) + 0.25000000\cos(t-3) \\ &+ 0.25000000\sin(t-s) - 2.25000000\cos(t+1) - 2.25000000\sin(t+1) \\ &+ 0.25000000\sin(t+3) - 0.25000000\cos(t+3) \end{aligned}$$

Neumann series approximation appears to converge.

This result does not clearly suggest how the solution behaves, but using the following input a graph can be produced giving a clearer indication of the solutions characteristics. The graph shown over the page was computed using thirty iterations.

```
> plot(Sol,t=-1..1);
```



This graph gives a clear indication that the result produced by the *fredneum* routine is close to the exact solution of $f(t) = t + 1.324841224\sin(t)$. The error produced by the routine when using thirty iterations is -9.4551×10^{-5}.

9 Program listing.

#Neumann series solution for Fredholm.

```
restart;
fredneum:=proc(G:algebraic,K:algebraic,lower:realcons,upper:realcons)
local i,iter,FuncT,Eqaution,converge,Conv;
#Check inputs.
if upper<lower then
ERROR(`Bounds on the integral are erroneous.`);
fi;
converge:=array(1..2);
iter:=10;
if nargs=5 then
if type(args[5],posint)=false then
ERROR(`variable for number of iterations must be of type (posint), using default
value.`);
fi;
if args[5]<3 then
ERROR(`Number of iterations must be three or more.`);
fi;
iter:=args[5];
fi;
```

```
FuncT:=G;
converge[1]:=0;
converge[2]:=FuncT;
#Neumann iterative series.
for i from 2 to iter do
FuncT:=simplify(eval(subs(t=s,FuncT)));
FuncT:=value(G+Int(FuncT*K,s=lower..upper));
if i=iter-1 then
converge[1]:=FuncT;
fi;
if i=iter then
converge[2]:=FuncT;
fi;
od;
Sol:=simplify(eval(subs(s=t,FuncT)));
if has(Sol,Int)=true then
ERROR(`Unable to evaluate integral.`);
fi;
#Print solution.
Equation:=G+Int(K*f(s),s=lower..upper);
print(`Equation entered:  `, f(t)=Equation);
print(`Solution of equation:  `, f(t)=Sol);
#Check for convergence.
Conv:=evalf(eval(subs(t=upper,converge[2]-converge[1])));
Conv:=sqrt(Conv^2);
if Conv<1/(upper-lower) then
print(`Neumann series approximation appears to converge.`);
else print(`Neumann series approximation does not appear to converge.`);
fi;
end;
```

Appendix Three.

Installation.

Installing the Volterra and Fredholm packages from a $3^1/_2$ inch floppy disc can be done by copying the entire contents of the disc into the /Maple directory using a file management program or simply through DOS. Once completed, run Maple and at the prompt type,

> read(`loadvolt.txt`);
> loadvolt();

and the Volterra package will be installed into Maple's library directory. If the error message *Unable to read file* is given, make sure Maple is reading from the /Maple directory and not /Maple/lib.

To install the Fredholm package, enter,

> read(`loadfred.txt`);
> loadfred();

All the routines listed in the previous two appendices can now be accessed, as can their help pages. A routine's help page can be displayed by entering the command,

> ?program_name;

where *program_name* is one of the routines, or either *volterra* or *fredholm*. The help pages for volterra and fredholm contain information on each package.
Example.

By entering the command

> ?code;

the help window on the following page is displayed.

FUNCTION: code - Conversion to ordinary differential equation form.

CALLING SEQUENCE:

code(g(t),k(t,s));

PARAMETERS:

g(t) - The forcing function, g, an algebraic expression in t.

k(t,s) - The kernel of the equation. An algebraic expression in t and s.

SYNOPSIS:

- code converts a linear Volterra integral equation of the second kind into it's equivalent ordinary differential equation form (If one exists).
- The routine will look for a conversion only up to a maximum of a tenth order differential equation. The circumstances for a Volterra equation to have an equivalent ordinary differential form of order four or more is extremely rare, if any exist at all.
- The reason why a maximum of a tenth order has been allowed is due to equations where the kernel has a power of ten or more, i.e. t^12.
- Once a equivalent ordinary differential equation has been found, the user may use the built in Maple function `dsolve` to find an expression for f(t). The constants or `initial values` for the `dsolve` command, may be found by typing `print(const);` which will display an array containing the found constants. The array has a maximum of ten entries, and in most cases only one or two will contain constants.
- For simplicity, the variables of the Volterra equation must be of the form s and t. This makes input more straightforward.
- This function should be defined by the command with(volterra) before being used.

EXAMPLES:

```
>with(volterra):

>>code(1,sin(t-s))
          Order of DE=2,  diff(f(t),t$2)=1

>print(const);
          [ 1 0 const[3] const[4] ]

# The array contains two constants, 1 and 0.
# i.e. f(0)=1, and D(f)(0)=0.

>dsolve({diff(f(t),t$2)=1,f(0)=1,D(f)(0)=0},f(t));
          f(t)=1/2 t^2 +1.
```

see also: voltlap, dsolve, D.

Bibliography.

[1] M.L. Abell, J.P. Braselton. *Differential equations with MapleV.* Academic press, inc. (1994).

[2] M. Abramowitz. *Handbook of mathematical functions with formulaes, graphs and mathematical tables.* Dover (1965).

[3] K.E. Atkinson. *A survey of numerical methods for the solution of Fredholm integral equations of the second kind.* SIAM (1976).

[4] C.T.H. Baker. *The numerical treatment of integral equations.* Oxford university press (1978).

[5] G. Birkhoff, G.C. Rota. *Ordinary differential equations, 4th ed.* Wiley (1989).

[6] W. Bolton. *Laplace and z-transforms.* Longman (1994).

[7] H. Brunner, P.J. van der Houwen. *The numerical solution of Volterra equations.* North Holland (1986).

[8] T.A. Burton. *Volterra integral and differential equations.* Academic press, inc. (1983).

[9] B.W. Char. *First leaves; A tutorial introduction to MapleV.* Springer-Verlag (1992).

[10] B.W. Char. *MapleV library reference manual.* Springer-Verlag (1991).

[11] B.W. Char. *MapleV language reference manual.* Springer-Verlag (1991).

[12] Courant and Hilbert. *Methods of mathematical physics, Vol. 1.* Wiley (1989).

[13] L.M. Delves, J. Walsh. *Numerical solution s of integral equations.* Oxford university press (1974).

[14] R. Kress. *Linear integral equations.* Springer-Verlag (1989).

[15] P. Linz. *Analytical and numerical methods for Volterra equations.* SIAM (1985).

[16] M.B. Monagan. *MapleV programming guide.* Springer (1996).

[17] R.E.A. *Advanced calculus.* R.E.A. (1991).

[18] F. Riesz, B. sz-Nagy. *Functional analysis.* Dover (1990).

[19] L.N. Trefethen, D. Bau III. *Numerical linear algebra.* SIAM (1997).

[20] F.G. Tricomi. *Integral equations.* Dover (1985).